Service-Oriented Ad Hoc Grid Computing

Dissertation

zur
Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem
Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

Thomas Friese
aus Marburg/Lahn

Marburg/Lahn 2006

Vom Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
als Dissertation am 17.10.2006 angenommen.

Erstgutachter: Prof. Dr. Bernd Freisleben

Zweitgutachter: Prof. Dr. Jörg P. Müller

Tag der mündlichen Prüfung am 23.10.2006

Abstract

Subject of this thesis are the design and implementation of an *ad hoc Grid infrastructure*. The vision of an ad hoc Grid further evolves conventional service-oriented Grid systems into a more robust, more flexible and more usable environment that is still standards compliant and interoperable with other Grid systems. A lot of work in current Grid middleware systems is focused on providing transparent access to high performance computing (HPC) resources (e.g. clusters) in virtual organizations spanning multiple institutions. The ad hoc Grid vision presented in this thesis exceeds this view in combining classical Grid components with more flexible components and usage models, allowing to form an environment combining dedicated HPC-resources with a large number of personal computers forming a "Desktop Grid".

Three examples from medical research, media research and mechanical engineering are presented as application scenarios for a service-oriented ad hoc Grid infrastructure. These sample applications are also used to derive requirements for the runtime environment as well as development tools for such an ad hoc Grid environment.

These requirements form the basis for the design and implementation of the *Marburg ad hoc Grid Environment* (MAGE) and the *Grid Development Tools for Eclipse* (GDT). MAGE is an implementation of a WSRF-compliant Grid middleware, that satisfies the criteria for an ad hoc Grid middleware presented in the introduction to this thesis. GDT extends the popular *Eclipse* integrated development environment by components that support application development both for traditional service-oriented Grid middleware systems as well as ad hoc Grid infrastructures such as MAGE. These development tools represent the first fully model driven approach to Grid service development integrated with infrastructure management components in service-oriented Grid computing.

This thesis is concluded by a quantitative discussion of the performance overhead imposed by the presented extensions to a service-oriented Grid middleware as well as a discussion of the qualitative improvements gained by the overall solution. The conclusion of this thesis also gives an outlook on future developments and areas for further research.

One of these qualitative improvements is "hot deployment" the ability to install and remove Grid services in a running node without interrupt to other

active services on the same node. Hot deployment has been introduced as a novelty in service-oriented Grid systems as a result of the research conducted for this thesis. It extends service-oriented Grid computing with a new paradigm, making installation of individual application components a functional aspect of the application.

This thesis further explores the idea of using peer-to-peer (P2P networking for Grid computing by combining a general purpose P2P framework with a standard compliant Grid middleware. In previous work the application of P2P systems has been limited to replica location and use of P2P index structures for discovery purposes. The work presented in this thesis also uses P2P networking to realize seamless communication accross network barriers. Even though the web service standards have been designed for the internet, the two-way communication requirement introduced by the WSRF-standards and particularly the notification pattern is not well supported by the web service standards. This defficiency can be answered by mechanisms that are part of such general purpose P2P communication frameworks.

Existing security infrastructures for Grid systems focus on protection of data during transmission and access control to individual resources or the overall Grid environment. This thesis focuses on security issues within a single node of a dynamically changing service-oriented Grid environment. To counter the security threads arising from the new capabilities of an ad hoc Grid, a number of novel isolation solutions are presented. These solutions address security issues and isolation on a fine-grained level providing a range of applicable basic mechanisms for isolation, ranging from lightweight system call interposition to complete paravirtualization of the operating systems.

Zusammenfassung

Die vorliegenden Dissertation behandelt den Entwurf und die Implementierung einer sogenannten *ad hoc Grid Infrastruktur*. Die Vision eines ad hoc Grids ist eine Weiterentwicklung des service-orientierten Grids hin zu einer robusteren, flexibleren und einfacher anzuwendenden Grid-Umgebung, die nach wie vor ihre Standardkonformität und Interoperabilität mit anderen Grid-Systemen bewahrt. Herkömmliche Grid-Middleware-Systeme konzentrieren sich auf eine Systemsicht, die sich dem "High Performance Computing" (HPC) verschrieben hat, in dem in einem Grid mehrere Rechner-Cluster oder Supercomputer zusammenfasst werden und der Zugriff auf diese Ressourcen auch über Organisationsgrenzen hinweg erlaubt wird. Die in dieser Arbeit vorgestellte Idee des ad hoc Grid Computing geht über diese klassische Sicht hinaus und kombiniert standardkonforme Grid Middlewarekomponenten, die auch in klassischen Grid-Systemen eingesetzt werden können, mit flexibleren Komponenten und Anwendungsmustern. Dies erlaubt es, eine verteilte Grid-Umgebung aus einer Kombination von dedizierten HPC-Ressourcen und einer Menge von aus Personal Computern (PC) aufgebauten "Desktop Grids" zu formen.

Als Einführung in das Anwendungsumfeld für eine solche service-orientierte ad hoc Grid Middleware werden zunächst drei Beispiele aus den Bereichen medizinische Forschung, Medienanalyse und Ingenieurwissenschaften skizziert. Von diesen einführenden Beispielanwendungen werden im Anschluss Anforderungen sowohl an eine Laufzeitumgebung für eine service-orientierte ad hoc Grid Middleware als auch die notwendigen Hilfsmittel zur Applikationsentwicklung in einem solchen Umfeld abgeleitet.

Die gewonnenen Anforderungen stellen die Grundlage für den Entwurf und die Implementierung des *Marburg ad hoc Grid Environment* (MAGE) und der *Grid Development Tools for Eclipse* (GDT) dar. MAGE ist eine Implementierung einer WSRF-konformen Grid Middleware, welche die in der Einleitung dieser Dissertation dargestellten Eigenschaften eines ad hoc Grids erfüllt. GDT erweitert die weit verbreitete Software-Entwicklungsumgebung *Eclipse* um Komponenten, die die Applikationsentwicklung sowohl für traditionelle service-orientierte Grid Middleware als auch für das ad hoc Grid vereinfachen. Im Rahmen dieser Arbeit wird damit der erste vollständig modell-getriebene Ansatz zur Grid-Service-Entwicklung und ein integrierter Ansatz zur Applikationsentwicklung und Ver-

waltung der Infrastruktur im Umfeld des service-orientierten Grid Computing präsentiert.

Abgeschlossen wird die vorliegende Dissertation durch eine quantitative Diskussion der durch die Erweiterung einer service-orientierten Grid Middleware eingeführten Mehraufwände sowie der dadurch gewonnen qualitativen Verbesserungen und einem Ausblick auf zuknftige Forschungsthemen im Umfeld dieser Arbeit.

Eine dieser qualitativen Verbesserungen stellt die Fähigkeit zur Bereitstellung (Installation und Entfernung) von Grid Services im laufenden Betrieb eines Rechnerknotens dar - das "Hot Deployment". Es wurde als Neuerung in service-orientierten Grid-Systemen als Resultat dieser Dissertation eingeführt. Hot Deployment stellt ein neues Paradigma für service-orientiertes Grid Computing dar, das die Installation von Applikationskomponenten zum funktionalen Bestandteil komplexer Grid-Applikationen macht.

Weiterhin verfolgt diese Dissertation die Idee der Verwendung von Peer-to-Peer (P2P) Netzwerk-Mechanismen für das Grid Computing und kombiniert zum ersten mal eine allgemein verwendbare P2P-Infrastruktur mit einer standardkonformen service-orientierten Grid Middleware. Im selben Umfeld wurden P2P-Netzwerke zuvor nur für die Auffindung von Datenreplikaten verwendet. Während viele Forschungsaktivitäten sich auf die Anwendung von P2P-Indexstrukturen für die Auffindung von Ressourcen konzentrieren, werden in dieser Dissertation auch die Kommunikationsfähigkeiten eines P2P-Netzwerkes zur Realisierung von barrierefreier Kommunikation eingesetzt. Obwohl die Argumente für problemlose Kommunikation durch standardisierte und akzeptierte Protokolle des World Wide Web für die ursprünglichen Web Service Standards gelten, entstehen durch die Anforderungen zur Zwei-Wege Kommunikation des WSRF-Standards spezielle Probleme, die durch diese Integration eines P2P-Netzwerkes gelöst werden können.

Herkömmliche Sicherheitsinfrastrukturen für Grid-Systeme konzentrieren sich auf die Sicherheit von Daten während der Übertragung und Zugangsbeschränkung in der Kommunikation zwischen Knoten des Grid-Systems. Diese Dissertation beschäftigt sich mit den innerhalb eines Rechnerknotens entstehenden Sicherheitsproblemen in einem dynamischen und veränderlichen service-orientierten Grid-System. Um den Bedrohungen zu begegnen, die von den dynamischen Fähigkeiten eines ad hoc Grids ausgehen, werden mehrere neuartige Isolationstechniken auf Service-Ebene für service-orientierte Grid-Systeme vorgeschlagen. Auf Basis dieser Vorschläge wurde eine Palette von Sicherheitslösungen entwickelt, die von sehr leichtgewichtigen Mechanismen - bezüglich ihrer Hauptspeicherverwendung - basierend auf dem Abfangen von Systemaufrufen bis hin zu eher schwergewichtigen Lösungen basierend auf kompletter Para-Virtualisierung des Betriebssystems beruhen.

Acknowledgements

I would like to acknowledge the help of many people during my studies that led to this thesis. In particular, I would like to thank Prof. Dr. Bernd Freisleben for supervising me and sharing his knowledge, insights and experiences in numerous discussions, and for his dedication as the research supervisor of the MAGE team encouraging me to contribute to the scientific community. I would also like to thank Prof. Dr. Jörg P. Müller for his support and guidance and the opportunity to participate in the ATHENA EU project, as well as the many discussions that helped further develop ideas of service-oriented computing in various application domains.

My thanks also go to Siemens AG, Corporate Technology, München, for supporting my work financially and Prof. Dr. Manfred Sommer for helping to initiate the cooperation between Siemens AG and the Philipps-University of Marburg. Within Siemens AG, Corporate Technology, I would like to thank the Peer-to-Peer research group at the Intelligent Autonomous Systems center for providing the RMF infrastructure and thoughts and impulses in an early stage of the service-oriented ad hoc Grid vision. I would particularly like to thank Steffen Rusitschka and Alan Southall for all the discussions we had.

I would also like to acknowledge financial support and input I have received for my work as part of the German Grid initiative (D-Grid) sponsored by the German ministry of research and education (BMBF), in the SFB/FK 615 "Media Upheavals" project sponsored by the German research foundation (DFG) and an IBM Eclipse innovation award.

I am very grateful to Matthew Smith for joining the work towards realizing the vision of service-oriented ad hoc Grid computing, for long and fruitful discussions, dedicated collaboration on our joint publications and the MAGE middleware implementation. I also want to thank the very knowledgeable and dedicated MAGE team for helping to implement, test and improve the system, namely Kay Dörnemann, Tim Dörnemann, Torsten Graf, Sergej Herdt, Steffen Heinzl, Ernst Juhnke, Markus Mathes, Elvis Papalilo, Christian Schridde, Matthias Weigand and Christian A. Wolf. I also thank the entire Distributed Systems Group at the Philipps-University of Marburg for their work and collaboration, especially Ralph Ewerth and Mechthild Keßler.

At the University of Siegen, Germany, I thank Julian Reichwald, Juniorprof. Dr. Thomas Barth and Prof. Dr. Manfred Grauer for the discussions and cooperation in the In-Grid project as part of the German Grid initiative (D-Grid) and the collaboration beyond this project.

I have also benefited from discussions with various members of the scientific community on several occasions and the comments given by many anonymous reviewers who provided feedback on the subjects published during the research in the context of this thesis.

I appreciate the patience of my family - my parents and Sylvia - and thank them for their support.

Contents

1

Introduction

The Grid computing paradigm [27, 70] is formed around the general goal to provide resources (e.g. computing power, data sources, special appliances, and even people) as easy as electricity is provided through the electrical power Grid. Similar ideas drive the closely related technologies of IBM's utility computing and on-demand business activities. Utility computing is defined to be *the on demand delivery of infrastructure, applications, and business processes in a security-rich, shared, scaleable, and standards-based computer environment over the Internet for a fee. Customers will tap into IT resources - and pay for them – as easily as they now get their electricity or water* [101]. Thus, the general idea is to provide services to a customer who can utilize those services on demand and gets charged for the usage of the service, e.g. in order to provide more complex business services to end users. In its current state, however, Grid computing focuses on the unification of resources by a middleware layer, to enable distributed computing within a fixed and preconfigured environment. Organizations or inter-organizational communities willing to *share* their computational resources typically create a centrally planned Grid, where dedicated Grid administrators manage the nodes and the offered Grid services. The installation of a production quality large scale Grid is far from trivial, making these administrators vital to the task.

If the Grid is to fulfill the vision of becoming the next-generation Internet (as described in [57, 69, 115]), the complexity of installation and maintenance must be reduced significantly. The Internet boom was made possible by making access to the Internet intuitive and transparent to the users. As a consequence, the number of users increased exponentially, which further increased the support for and the acceptance of the new medium. Grid middleware supporting such a massive adoption of the Grid paradigm must hold many properties of an on-demand infrastructure as stated in [40]: it must be *sharable* among different

users, provide *standardized* services and communication protocols and it must be *flexible* and *scalable*.

The introduction of the service-oriented computing paradigm and the corresponding web service standards such as WSDL [45] and SOAP [171] in the field of Grid computing through the Open Grid Services Architecture (OGSA) [68, 71] is a major step towards reducing the complexity of Grid use, operation and maintenance. While OGSA describes the higher-level architectural aspects of service-oriented Grid computing, the Web Service Resource Framework (WSRF) [132] is a fine-grained description of the infrastructure required to implement the OGSA model. Service-oriented Grid computing offers the potential to provide a fine grained virtualization of the available resources to significantly increase the versatility of a Grid. It can be employed to create a broader user base as a catalyst for new Grid developments by extending the initial vision of the Grid - connecting the world's supercomputing centres - to also incorporate the much larger community of desktop computers. The influx of users will allow the Grid to offer the possibility of harnessing the unused CPU cycles (or other resources) of idle workstations, as found in practically every organization, by combining them on demand to spontaneously form an *ad hoc Grid* without a preconfigured, fixed infrastructure.

To achieve this goal, a number of new challenges must be taken into account. For example, through the extension of the Grid by non-dedicated resources, the complexity of the Grid is greatly increased. When a large number of nodes are added to the Grid on a dynamic basis, central administration is no longer feasible. The heterogeneity of the system is increased and the reliability of the nodes is decreased due to reboots or crashes caused by the regular users of those nodes. The system itself must be capable of coping with the dynamic topology changes of the underlying network and the heterogeneity of the nodes to form an ad hoc Grid autonomously. Security is also of vital importance to such an extended Grid system. Since the number of users within a system is increased, new security mechanisms are needed to ensure that malicious code cannot harm legitimate services running on the Grid.

The previously described challenges focus on the runtime complexity of enabling ad hoc Grid computing. A middleware addressing all of these challenges can support Grid applications in a spontaneously formed environment that copes well with frequent changes in the infrastructure. However, the vision of ad hoc Grid computing is not limited to reducing the complexity of creating and maintaining a runtime environment that supports Grid applications. An ad hoc Grid environment must also foster *fast and easy development* of new Grid applications in order to enable application of the Grid computing paradigm in more application domains.

Application development for distributed systems is, in general, a complex task. The inherent complexity of existing Grid middleware systems further increases the barrier felt by non-experts in distributed application development when try-

ing to apply the Grid computing paradigm to their domain of expertise. A flexible and autonomous middleware system together with a flexible programming model that is easy to understand and helps developers rather than overwhelming them with functionality can partially help to achieve this goal of fast application development. A well integrated development environment that makes use of the underlying ad hoc Grid middleware concepts and considers the general goal to provide development support both for Grid middleware experts and experts in various application domains that do not hold intimate knowledge of distributed systems development or Grid middleware is a further step towards true ad hoc Grid computing, lowering the barrier of adopting the Grid paradigm by a broader user community.

In this thesis, the main problems involved in realizing a service-oriented ad hoc Grid to provide computing resources to every participant on demand are presented and solved. The middleware solutions to these problems are based on peer-to-peer node and service discovery, hot deployment and administration of services into a running system without disrupting other services already running there, added inter-service security by ensuring that each service runs within its own sandbox and has no direct access to the running code of other services, and a flexible trust management system. The important parts of a prototypical implementation based on the Globus Toolkit 4.0 (GT4) will be described. The service-oriented ad hoc Grid environment introduced in this paper opens up a whole new range of resources to be tapped and expands the potential user base of the Grid paradigm significantly.

An ad hoc Grid is a spontaneous formation of cooperating heterogeneous computing nodes into a logical community without a preconfigured fixed infrastructure and with only minimal administrative requirements [156, 83]. The vision of on-demand provisioning of application services as a utility still allows for different hosting models such as *collocated*, *dedicated* and *shared* hosting [128] that can rely on tedious manual maintenance of a preconfigured infrastructure. The ad hoc Grid extends the on-demand idea to the application infrastructure itself. With only limited installation overhead, the logical community formed by an ad hoc Grid middleware is able to offer most of the basic infrastructure services required to provide services as a utility to a large and changing set of users. The number of non-dedicated Grid nodes in such an ad hoc Grid is much higher than in traditional Grid systems, demanding non-intrusive operation of the ad hoc Grid middleware.

Thus, the view of an ad hoc Grid environment in this work goes beyond the preconfigured, dedicated Grid infrastructures existing today to encompass frequent dynamic additions of computational resources to the Grid. This includes workstations within organizations as well as scattered personal computers, similar to the basic idea of many distributed computing projects like SETI@Home [108].

Figure 1.1 shows how two separate ad hoc Grids are composed. The first ad hoc Grid (A) spans two organizations; the second (B) is created from scattered

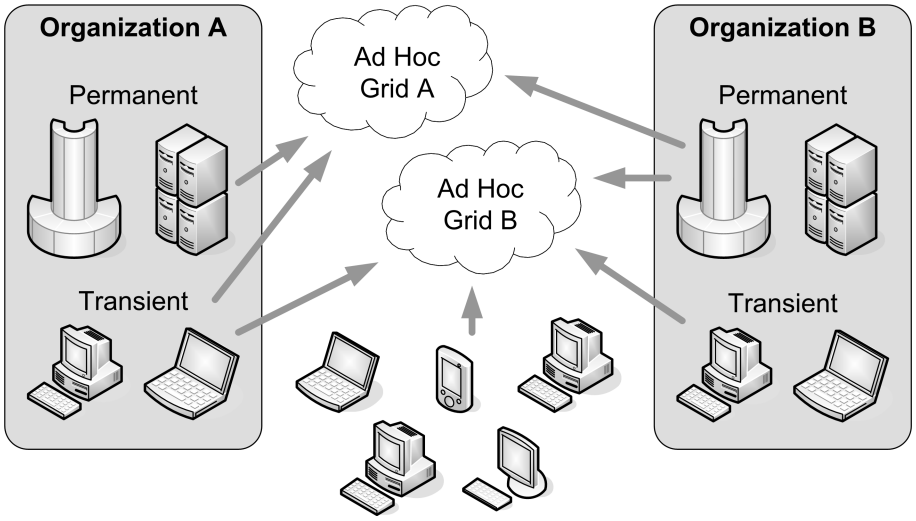


Figure 1.1: Ad Hoc Grid architecture overview.

nodes on the Internet. Both Grid communities form a virtual organization using the existing Internet infrastructure.

While ad hoc Grid A encompasses transient nodes (e.g. non-dedicated workstations), it also includes dedicated high-performance computers. In contrast, ad hoc Grid B is made up solely of transient individual nodes. While ad hoc Grid A bears a greater resemblance to traditional Grid systems, ad hoc Grid B illustrates the shift to a personal Grid system, built without the resources of a large organization.

The spontaneous formation of a Grid system also complies with the on-demand or utility computing paradigm pushed by IBM [143]. In this scenario, computational peak loads are outsourced to organizations offering computational power or the required resources. For this, the possibility of renting resources dynamically with only minimal administrative overhead is desirable. Thus, it must be possible to acquire and configure the needed resources without requiring the administrator to manually facilitate each and every transaction.

A number of different user groups - sometimes the same users in different roles - are involved in creating, maintaining and using a Grid system. *Resource providers* are the owners of computational nodes or other physical resources that form the basis for spanning a Grid system. *Solution producers* are the owners of software solutions or databases which are deployed on the resource providers' assets in the Grid. In this sense, the group of solution producers include both the *solution developers* who create a particular piece of software representing special algorithms and knowledge and also the *solution providers* who install and maintain software in the Grid system and offer them as an asset to applications and their users. The last group of stakeholders in a Grid system are the *Grid users*. Again, this term may be used for a wide range of parties involved in the

usage and lifetime of a Grid. As a very basic definition, the group of Grid users can be defined as the owners of input data to a solution producer's product.

This view on the users of a Grid system acknowledges the involvement of the owners or producers of physical resources, software and the input data used in Grid applications. The collection of those resources into a Grid system requires the availability of Grid middleware provided by *middleware producers*. For some considerations regarding, for example, security and trust, it is reasonable to distinguish this last group from the solution producers even though middleware components can also be regarded as basic solution components.

Some of the components of the system presented in this work are specifically geared towards supporting the interaction of these different user groups joined in an ad hoc Grid environment. The key view of the users on such a Grid environment is a service-oriented view. Users and developers interact directly or indirectly through higher level applications with Grid services that represent the functional components of the environment and virtualize the underlying resources such as the raw computational power of the nodes involved in the interaction. This view on the service-oriented Grid goes farther than the current middleware developments that often only use the web service protocols for seamless interaction with a more traditional view on the Grid as a preconfigured and static collection of high performance computing resources that are accessed through their work queues.

The major research contributions of this thesis are:

- The general idea of developing a service-oriented ad hoc Grid environment was first proposed in 2004 in several publications the thesis is based on. In particular, the ability of offering hot Grid service deployment (i.e. non-intrusive installation and removal of Grid services on a Grid node) was introduced as a novelty in a service-oriented Grid computing environment as a result of this work in 2004. Later on, the initial implementation was provided to the members of the Globus project team at the University of Chicago and the Argonne National Laboratory, and hot Grid service deployment is currently being integrated into future releases of the Globus toolkit. The hot Grid service deployment functionality represents a new paradigm in service-oriented Grid computing by offering component distribution as a primitive for the development of complex Grid applications.

- This thesis pursues the general idea of using peer-to-peer (P2P) networking mechanisms for Grid computing by presenting the first integration of a general purpose P2P computing infrastructure into a standards-compliant service-oriented Grid middleware. P2P networking has only been applied for replica location of experimental data in the same environment before. While many research efforts focus on the application of P2P index structures for discovery purposes, this thesis also taps the communication capabilities of the P2P network to enable seamless communication over network

partitions in an Internet environment. Although the argument for seamless communication through standard world wide web protocols holds for original web services, the two-way communication nature of the WSRF standards poses special problems addressed by this integration.

- While traditional Grid security infrastructures focus on message level security and access control for inter-node communication in a service-oriented Grid environment, this thesis focuses on the inherent intra-node security threats for a dynamic and changing Grid community. To counter the threats arising from the dynamic capabilities of an ad hoc Grid, a number of novel service-level isolation techniques are introduced in the field of service-oriented Grid middleware. A wide range of security solutions, ranging from very lightweight approaches with respect to memory utilization through operating system call interposition to rather heavyweight approaches such as complete operating system para-virtualization, have been developed.

- As an inherent element of service-oriented ad hoc Grid computing, not only the runtime environment, but also sophisticated development support must be addressed. This thesis presents the first model-driven approach to Grid service development and developer and administrator support in an integrated environment in the service-oriented Grid computing domain. Very positive feedback from a growing external user community suggests that such tools are helpful to lower the entry burden into service-oriented Grid computing and to allow user-friendly development of complex Grid applications. A novel separation of one of the modeling layers - the platform specific layer - into an upper, application specific and a lower, target system specific sub-layer is introduced, allowing to easily target different concrete implementations of the same high-level middleware paradigm.

The following papers have been published as part of the research conducted in the context of this thesis:

1. T. Friese, B. Freisleben, S. Rusitschka, and A. Southall. A Framework for Resource Management in Peer-to-Peer Networks. In *Proceedings of the International Conference NetObjectDays*, LNCS 2591, pages 4–21, Erfurt, Germany, 2002. Springer-Verlag.

2. M. Smith, T. Friese, and B. Freisleben. Towards a Service-Oriented Ad Hoc Grid. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing*, pages 201–209, Cork, Ireland, 2004. IEEE Press.

3. T. Friese, M. Smith, and B. Freisleben. Hot Service Deployment in an Ad Hoc Grid Environment. In *Proceedings of the 2nd International Conference on Service-Oriented Computing*, pages 75–83, New York, USA, 2004. ACM Press.

4. M. Smith, T. Friese, and B. Freisleben. Intra-Engine Service Security for Grids Based on WSRF. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing and Grid*, pages 644–653, Cardiff, UK, 2004. IEEE Press.

5. R. Ewerth, T. Friese, M. Grube, and B. Freisleben. Grid Services for Distributed Video Cut Detection. In *Proceedings of the 6th IEEE International Symposium on Multimedia Software Engineering*, pages 164–168, Miami, USA, 2004. IEEE Press.

6. T. Friese, J. P. Müller, and B. Freisleben. Self-Healing Execution of Business Processes Based on a Peer-to-Peer Service Architecture. In *Proceedings of the 18th International Conference on Architecture of Computing Systems*, LNCS 3432, pages 108–123, Innsbruck, Austria, 2005. Springer-Verlag.

7. E. Papalilo, T. Friese, M. Smith, and B. Freisleben. Trust Shaping: Adapting Trust Establishment and Management to Application Requirements in a Service-Oriented Grid Environment. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing*, LNCS 3795, pages 47–58, Beijing, China, 2005. Springer-Verlag.

8. T. Friese, J. P. Müller, M. Smith, and B. Freisleben. A Robust Business Resource Management Framework Based on a Peer-to-Peer Infrastructure. In *Proceedings of the 7th International IEEE Conference on E-Commerce Technology*, pages 215–222, Munich, Germany, 2005. IEEE Press.

9. M. Smith, T. Friese, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications, In Barth, T., Schüll, A. (eds.): *Grid Computing*, Vieweg Verlag, pages 191–210, 2006.

10. T. Friese, M. Smith, and B. Freisleben. The Service-Oriented Ad Hoc Grid, In Barth, T., Schüll, A. (eds.): *Grid Computing*, Vieweg Verlag, pages 143–191, 2006.

11. M. Smith, T. Friese, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications. In *Proceedings of the International Conference on Internet and Web Applications and Services*, pages 139–146, Guadeloupe, French Guyana, 2006. IEEE Press.

12. T. Friese, J. P. Müller, and B. Freisleben. Integrating Peer-to-Peer Technology into a Web Service Environment. In *Multikonferenz Wirtschaftsinformatik, Workshop: Betriebliche Anwendungen des P2P und Grid Computing*, Passau, 2006.

13. M. Smith, T. Friese, M. Engel, B. Freisleben, G. Koenig, and W. Yurcik. Security Issues in On-Demand Grid and Cluster Computing. In *Proceedings of*

the 6th IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW'06), pages 24–37, Singapore, 2006. IEEE Press.

14. M. Smith, T. Friese, M. Engel, and B. Freisleben. Countering Security Threats in Service-Oriented On-Demand Grid Computing Using Sandboxing and Trusted Computing Techniques. *Journal of Parallel and Distributed Computing*, pages 1189–1204, 2006.

15. T. Friese, M. Smith, and B. Freisleben. GDT: A Toolkit for Grid Service Development. In *Proceedings of the 3rd International Conference on Grid Service Engineering and Management*, pages 131–148, Erfurt, Germany, 2006.

16. M. Mathes, S. Heinzl, T. Friese, and B. Freisleben. Enabling Post-Invocation Parameter Transmission in Service-Oriented Environments. In *Proceedings of the International Conference on Networking and Services*, pp. 55, Silicon Valley, USA, 2006. IEEE Press.

17. S. Heinzl, M. Mathes, T. Friese, M. Smith, and B. Freisleben. Flex-SwA: Flexible Exchange of Binary Data Based on SOAP Messages with Attachments. In *Proceedings of the IEEE International Conference on Web Services*, pages 3–10, Chicago, USA, 2006. IEEE Press.

18. T. Friese, M. Smith, B. Freisleben, J. Reichwald, T. Barth, and M. Grauer. Collaborative Grid Process Creation Support in an Engineering Domain. In *Proceedings of the International Conference on High Performance Computing*, Bangalore, India, 2006 (to appear). Springer-Verlag.

This thesis is organized into 7 chapters.

Chapter 1, gives a general introduction into the topic of service-oriented ad hoc Grid computing and motivates the proposed change to the existing service-oriented Grid computing paradigm towards a more usable ad hoc Grid.

Chapter 2 presents the requirements of service-oriented ad hoc Grid computing. These requirements are derived from three sample applications from the medical research, media analysis and mechanical engineering domains.

Chapter 3 introduces a discussion of related work in the field of service-oriented Grid computing and its transition towards service-oriented ad hoc Grid computing. Related work is selected and structured according to the problems addressed in this thesis and the solutions presented throughout this thesis.

Chapter 4 discusses the design of a service-oriented ad hoc Grid middleware. Fundamental design decisions are presented for both the runtime components of such a Grid middleware and for a development environment supporting fast development of applications for the service-oriented ad hoc Grid.

Chapter 5 describes an implementation of the middleware and development and management tools. The distinct consideration of a runtime infrastructure and

a development and management support environment is followed throughout the entire thesis.

Chapter 6 evaluates the runtime and the development and management environment of the service-oriented ad hoc Grid middleware. This chapter contains measurements as well as a qualitative analysis of the system and a sketch of the realisation of the sample applications using the middleware and tools presented in this thesis.

Chapter 7 concludes the thesis with a summary and outlook on future work.

2

Problem Statement

2.1 Introduction

This chapter introduces three different application scenarios for a service-oriented ad hoc Grid computing. The three applications come from the medical and mechanical engineering application area as well as from media research. They present use cases where a service-oriented ad hoc Grid could improve the situation of individuals with a high demand for collaborative use of distributed resources.

After the presentation of the sample application scenarios, several requirements for an ad hoc Grid middleware solution are presented. In addition to the requirements of the runtime components of such a middleware, considerations for the development of applications on top of this infrastructure are presented. The requirements presented in this chapter are derived mainly from the sample applications introduced in the first part of the chapter. Additional requirements for a service-oriented Grid may exist, but they are not discussed in the scope of this work.

2.2 Sample Applications

2.2.1 Medical Application

The medical application described as a usage scenario for the ad hoc Grid is a data processing application in sleep research and sleep medicine as part of the "MediGrid" project, part of the German "D-Grid" [50] research program. The general objective of this medical research area are sleep and sleep disorders as well as the development of clinical diagnosis and therapy of sleep-wake disorders.

In sleep medicine, measurements of various body functions of a patient are measured throughout a sleeping period. Those measurements include the electrocardiogram (ECG), breathing activity and the patients' brain activity measured as the electroencephalogram (EEG). In a first step, the different measurements need to be evaluated in order to reduce them to a sleeping protocol that describes the sleeping cycle of a patient as an order of different sleep phases with respect to a multi-level sleep classification system. Today, this classification is mostly done manually even though classification algorithms exist that can automatically deduce a sleeping protocol from a given sleeping cycle data record. This automatic deduction requires the application of various filter functions and transformations on the experimental data. The resulting sleep protocol can be used to help physicians to deduce sleep disorders and possibly connect them with other diseases caused by the sleep disorders.

The work of medical researchers and physicians in a hospital can be supported by using ad hoc Grid technology to form a Grid environment using their desktop personal computers for the processing of such patient data records. The individual data sets are of reasonable size ($<$ 500 MB per night and patient) to apply different filters and transformations to the data distributed over the local hospital network.

A second use case from the same field considers larger collections of patient records for medical research. There is hope in the medical research community that the clustering of a set of patient sleeping records will show a correlation between patterns in the measurements and certain diseases. The goal of the researchers is the discovery of a discriminating function that can be used to classify a patient regarding his or her risk to develop a particular disease from the patterns found in his or her sleep activity.

The ad hoc Grid can be especially useful in this second application scenario. Medical researchers demand an application that lets them apply a chain of filters and transformations on a large collection of data records. The individual filters and transformations as well as their overall combination are a topic of research undergoing constant development and changes. The ad hoc Grid can support this use case with its inherent capability to dynamically deploy new components and processes into an infrastructure that provides the required computational resources.

2.2.2 Media Analysis Application

The media analysis application scenario presented in this section is motivated by a large media research project currently conducted at the University of Siegen, University of Marburg, University of Dortmund and the FHG St. Augustin, Germany, entitled "Media Upheavals". It is aimed at investigating the foundations and the structural aspects of the comprehensive media upheavals and their impact to media culture and media aesthetics at the beginning of the twentieth century

(introduction of films and cinema) and in the transition to the 21st century (Internet and WWW). The sample application in this scenario should provide a high-performance video content analysis system to support other projects in applying film analysis. The software workbench Mediana [61] is currently under development to provide such support. In addition to database tools, Mediana includes several algorithms for video content analysis, including cut and shot boundary detection, text detection and text segmentation, camera motion estimation, and face detection. Although the processing power of today's computers is enormous, the processing time of such algorithms and the data-intensive multimedia file organization are still challenging issues.

In this application scenario an ad hoc Grid middleware can be used to implement the computationally demanding analysis application as a distributed back end application for the Mediana workbench. Different media scientists use the Grid infrastructure to combine their computational resources, but also to join their pools of multimedia as well as feature data. Additionally, the communications facilities of the Grid infrastructure can be used to implement direct and synchronous collaboration functionality to allow geographically distributed media scientists to collaborate in developing and testing hypotheses in their area of interest.

Verification of a hypothesis from a media analysis point of view often requires the combination application of different feature extraction algorithms on a large collection of multimedia data. On a high level, the media scientists may combine existing services for feature extraction into more complex processes to be applied to the raw data set or previously extracted features. Computer scientists provide new feature extraction algorithms as Grid services that are in turn deployed to the nodes of the Grid network. Together, they realize the high level functionality provided in the Mediana workbench. For more details about a first Grid based implementation of a video cut detection application, the reader is referred to [60].

2.2.3 Engineering Application

In this section, the engineering process for the creation of a metal casting process model is presented as a sample application. This sample application is a use case under investigation in the context of the "In-Grid" research project, a community Grid project for engineering applications, part of the German "D-Grid" research program. Casting is a sub-domain of metal forming. Only a simplified view on the engineering process is presented, which does not reflect the entire complex field of metal forming. For more information regarding the complexity involved in collaborative engineering particularly in the field of metal forming and casting, the reader is referred to e.g. [127, 178].

In the sample scenario, a casting engineer is assigned to develop a casting process for the production of a part made from cast metal. Creation of the casting process involves optimization of the geometry of the final part and the mold as

well as the parameters of the production process. The casting process model development cycle starts with a problem definition, expressed by the casting engineer and progresses through some iterations of model definition, simulation and refinement. Initially, a given problem definition by a customer is modeled as a casting process model by a numerical simulation expert. The numerical simulation expert periodically discusses the evolution of the initial model with the casting engineer during the design phase. Both experts have to combine their expertise to successfully define an accurate model for the casting process. To verify the accuracy of the resulting model, it is typically numerically simulated and compared to knowledge about real casting processes held by the casting engineer. If this first simulation run does not match reality, the model needs to be calibrated and further model variants are created by the simulation expert and the casting engineer until the model satisfactory reflects a real casting process.

When a model is calibrated, the optimization of the model begins by automatically generating a number of n new models by varying the parameters in the casting process model. Those model variants can be evaluated in parallel, and the results from the simulation runs flow back to the optimization algorithm. This procedure iterates until the optimized casting process meets the requirements set by the casting engineer.

The benefit of simulated prototyping is constrained by the accuracy of the simulation environment. Both the creation and use of the simulation application require great expertise in the metal casting domain. Furthermore, applying numerical simulation for this purpose introduces an extremely high demand for computational capacity since a single - sufficiently precise - simulation run typically lasts several hours up to days. Since many small and medium sized engineering enterprises are not capable of acquiring and maintaining high performance computing resources, outsourcing of computational demanding tasks is necessary. Grid computing promises to offer the infrastructural components to realize this outsourcing activity as easy as plugging into the electrical power Grid.

Additionally, the expertise of specialists from different domains (casting, numerical simulation, Grid experts, customers) is required to achieve the desired results. Especially small and medium sized companies cannot afford to employ all the various experts; rather close collaboration of different, geographically distributed experts is required for many engineering applications similar to the casting example introduced in this section.

The previously described engineering process is only a part of a larger internal process of both the customer and the engineering company. It is also common that the model development process has to be slightly modified for other cases where a casting model must be developed.

2.3 Requirements for an Ad Hoc Grid Solution

2.3.1 Runtime Environment

The runtime system of an ad hoc Grid middleware is the software component that bridges the gap between applications and the underlying infrastructure. The idea of providing such an application infrastructure spontaneously presents some special demands that are usually not met by regular service-oriented Grid middleware platforms. In the following, these requirements are discussed on the platform level.

Node Communication

Even though the web service protocols have been designed to take advantage of well known and established Internet standards, some communication barriers exist on the Internet that pose a problem for an ad hoc Grid environment. Private networks are hidden behind firewalls and routers performing network address translation. Nodes behind these barriers can work perfectly well as clients that consume Grid service functionality, providing services from within those confined network realms requires manual configuration effort. Approaches to support automatic traversal of such barriers are required.

Node and Service Discovery

In an ad hoc Grid environment, the network topology is dynamic (i.e. rebooting of workstations, movement of laptops, replacement of computers) and thus a node detection solution geared towards frequent node arrivals and departures is required. While arrival or departure of nodes should be discovered as quickly as possible, a balance needs to be found between keeping the topology information up to date and flooding the network with discovery messages.

OGSA [68, 71] and WSRF [132] define virtualization of available resources at the system-independent level of resource access to allow uniform access to a heterogeneous system. The hardware and operating system or underlying implementation of a service is hidden from the caller of a service. Support of the standard Grid service interfaces is guaranteed and sufficient to allow access to the resource. Even the instantiation of a deployed Grid service is system-independent, since it is specified to be handled by a gatekeeper (service factory).

To enable autonomous deployment of services, meta-information from the Grid node must be available to the deployment service, so it can reliably operate in a heterogeneous environment. While the underlying middleware of each node is guaranteed to support the Grid service interface, the way it implements this is not specified by OGSA or WSRF. When deploying services to newly discovered nodes, it is, however, necessary that the service implementation is supported by the Grid platform of the node. Information about the underlying hardware and

operating system is particularly important when deploying legacy code, because tight integration of system resources is a common occurrence there.

Typical information needed is operating system type, available hardware resources and availability of required libraries. Furthermore, information on the reliability of the nodes can be taken into account when services are to be deployed, so nodes with long up times are given priority over nodes which frequently reboot or crash.

Furthermore, it must be possible to discover already deployed services based on service names or service descriptions in order to use the services.

The demand for a spontaneous service and node discovery can be clearly motivated in the medical as well as the media analysis sample application scenarios. In both cases, the resources forming the Grid infrastructure are assumed to be provided collaboratively by the application users. There is no clear separation between resource providers and resource users and therefore no clear responsibility for the installation and maintenance of a registry can be identified. Resource providers in the engineering example would benefit from an automatic discovery component since it alleviates the management overhead for the infrastructure, even though dedicated infrastructure providers for computational resources might be willing to also maintain registry components.

Service Deployment and Administration

The invocation of services on various nodes in a Grid system is only possible if they are available on those nodes. For a large scale ad hoc Grid, the time consumption for manual deployment is prohibitive, and management is difficult due to the fluctuating availability of the nodes. Even with the availability of advanced Grid programming toolkits, deployment of services has been identified as a critical issue [89]. Furthermore, the number of Grid services will steadily rise with the number of users, further increasing the management cost of the Grid environment. In a dynamically changing environment, deployment is even more critical, since there is no single deployment cycle that reaches all machines. Instead, services need to be deployed and instantiated on demand on machines as they become available.

Service deployment becomes part of an ad hoc Grid application instead of being handled by a system administrator as a precondition to the use of a service. Instead of only providing predefined services, the computational nodes of the bare Grid become a resource in themselves. This resource can be tapped by applications using the hot deployment service to leverage spare resources into their computational group. When a node becomes available that meets the requirements for the deployment of a service, the application can autonomously carry out the deployment and use the newly available machine for its application flow.

In a production environment, every operation needs to be non-intrusive, i.e.

it does not interfere with the execution of other services already running on the Grid. The ability to introduce or remove a service without interruption of other operations is vital for the vision of a highly flexible ad hoc Grid.

Non-intrusive service deployment and management is required in all sample application scenarios. The medical and media use cases explicitly include the need to deploy newly developed component services for the verification of research hypotheses. The engineering application requires a solution fitting to a production environment and requires frequent customization of the overall process and also its component services. In all cases, a guarantee must be given that management operations on component services is non-intrusive to the applications of other users in the ad hoc Grid environment.

Service Security and Trust

Security is a major aspect in all distributed systems, since there is always the possibility that a node introduces malicious code. In an ad hoc Grid, several new aspects must be dealt with beyond the standard security requirements existing in previous Grid systems. In traditional systems, installing a service requires administrative privileges on a Grid node allowing the operations. Services usually can only be installed by a very small number of people and trust can be assumed between all parties. In a large ad hoc Grid, on the other hand, it is possible that users unknown to each other operate on the same node. This gives rise to new security issues. One major new security threat is that a trusted node is running further unknown services. Here, inter-service security must be offered, since fair play is not guaranteed any more. This includes strong compartmentalization of the process spaces in which the individual services are executed.

The previously described security considerations are of great importance to all presented application scenarios, but they are especially strong in the medical and engineering use cases. For the medical use case, there are even legal regulations governing the acceptable handling of patient records. In the engineering application, the resulting casting process model is of great strategic value to the customer of the casting engineer. An ad hoc Grid infrastructure must guarantee strong separation from competitors using the same resources to be accepted by users in this domain. The security requirements in the media analysis use case are not as strict as for the medical and industrial application.

Data Handling

Many Grid applications require the transmission of large amounts of data. In the three sample scenarios presented in section 2.2 the data sets that need to be transmitted to a Grid service are the sleep data records, large video files for the media science application and geometry models as well as simulation output for the engineering application.

Embedding such data in the SOAP messages used for service invocation is not a reasonable approach, because the XML formats that SOAP is built on are not suitable to hold large binary objects. This fact has been addressed by the introduction of the *SOAP Messages with Attachments (SwA)* [25] specification. It defines how a SOAP message and several binary objects can be composed in a Multipurpose Internet Mail Extensions (MIME) message of type multipart/related. Each message part is separated by a unique delimiter string defined at the beginning of the message [75].

Unfortunately, MIME does not allow random access to arbitrary message elements without prior reception of the entire message. A Grid service cannot decide which binary objects to receive and it cannot decide the order in which binary objects are transferred since each SwA message is transferred as a whole using the underlying transfer protocol (in most cases HTTP).

A common implementation pattern to be found in applications that require the transmission of large amounts of binary data, is the transmission of data location pointers (URIs or more complex reference structures) that are interpreted in the application logic of the service and that are also application specific. The service implementation then uses other data access components, e.g. OGSA-DAI [18, 106] or RFT [170] in Grid environments. A drawback of such solutions is that they are often not very flexible and require the availability of fixed infrastructural services such as a GridFTP server.

For the ad hoc Grid, a flexible solution for data handling is required that integrates well with available fixed components in a Grid infrastructure as well as the components found in an ad hoc Grid environment. From a development point of view, the handling of data transmissions should not be a concern of Grid application developers, rather a Grid middleware should provide strong architectural support for various data transmission patterns that allow developers to create complex applications easily.

Grid Process Execution

Different demands from different user groups determine the requirements for a Grid process execution solution. Service-orientation brings the vision that Grid services can be used as the basic building blocks for more complex applications. These applications should be created by developing a description of the overall workflow to carry out based on the basic services. The abstract description of this workflow is referred to as a Grid process, which is a blueprint for a concrete workflow on a Grid system that in turn needs to be executed and controlled, for example, by a Grid process execution engine or a distributed process execution mechanism.

High level composition of Grid processes also offers the ability for non-programmers to develop Grid applications from their high level point of view. Process execution in that respect is a means of abstraction from the underlying

middleware. This, on the other hand, requires abstract constructs in the process description language that captures the semantics of the functionality available from the Grid middleware. In case of the ad hoc Grid, the most important features that need to be reflected in the process description language, are constructs for parallel execution of activities on many nodes and support for the dynamic nature of the underlying infrastructure that is spontaneously formed and potentially changes rapidly during the runtime of a Grid process. Reasonable support for resilience of the process execution against failure of individual basic activities must be provided, especially in the ad hoc Grid where a high churn rate in connected nodes makes failures of basic activities very likely. A Grid process execution environment also needs to handle a possibly large number of concurrent basic service invocations and must be implemented to provide sufficient scalability.

Business process execution has become very popular, automating other high level economic processes within an enterprise. Grid process execution technology should easily integrate with business process execution. In addition to allowing seamless integration into the internal and external business process of an enterprise, the tools and technologies developed for the creation and management of the business processes are ideally also applicable to the Grid processes used in an enterprise environment.

All three sample applications bear a strong need for flexibility in the specification and execution of high level application workflows. A process execution system for the service-oriented ad hoc Grid environment that incorporates support for concurrency and other offerings of the underlying middleware can greatly improve flexibility in the development of the applications described in section 2.2.

2.3.2 Development and Management Environment

From an application development point of view, middleware does only bridge the gap between an application and the set of physical or logical resources. It can ease application development by hiding the heterogeneity of the various operating system and hardware platforms included in a typical Grid system. However, application development using this application programming interface to the middleware is clearly targeted at specialists trained in software development.

Typical usage scenarios for a classical Grid system are scientific applications that require massive amounts of computational power or large scale data storage to conduct numerical simulation and data analysis. Simulations or individual data processing steps are usually only part of a larger, more complex application workflow. Users from various application domains such as high energy physics, astrophysics or engineering use Grid resources to solve concrete problems. The functionality required to solve such a problem is typically not provided by a standard application, there is rather a requirement for customization of an application to meet the users' requirements or even the development of a completely new application. Based on the ad hoc Grid idea of instant availability of a Grid

infrastructure, it is even more important for the ad hoc Grid than for classical Grid middleware to support fast application development. To provide the best possible support to developers, an integrated development environment should be provided by an ad hoc Grid environment that supports rapid application creation.

Component and Process Creation

In a typical Grid environment, the Grid user is directly involved in the development of the applications that help solving a given problem, as they are typically complex problems that require expertise in the application domain. Again, this leads to a shifted role of the Grid user in the ad hoc Grid environment. On a high level of abstraction, shielded from the inherent complexity of the underlying Grid middleware, the Grid user should be enabled to design an application as a solution to his or her problem. This application can be the composition of several basic applications or services into a more complex application. This compositional approach to application development is perfectly supported by the shift of the Grid computing paradigm to the adoption of service-orientation. To further particularize the basic requirement for fast application development support by an ad hoc Grid infrastructure, this development support must enable fast *creation of basic services* as well as support for the *composition of basic services into more complex applications* that fully leverage the underlying Grid infrastructure.

Separation of Concerns and Abstraction

Another requirement stemming from the involvement of many developers with expertise in different domains of interest is to support *separation of concerns* during the development of an application. For every expert working on a separate problem of a Grid application project, it should be possible to concentrate on his or her aspect of development (e.g. the high level application logic or the integration with the Grid middleware) while still producing a joint solution. The different concerns of the different groups of users involved in the development of the application should also be supported by offering the ability to work on *different levels of abstraction* and hide the complexity of the underlying Grid middleware. If details of the final application are of no concern to an application domain expert, an integrated development solution for the ad hoc Grid should offer an abstracted view on the final application that allows such users to work on the application without being distracted by too much detail.

Collaboration

A very basic idea of Grid computing is the collaboration of organizations and individuals by sharing their resources with each other and directly work together in a common application environment. This collaboration usually even involves real time communication via voice or video conferencing and direct steering and

control of the Grid application used to solve a particular problem. *Collaboration between different experts* must also be facilitated by a development environment. This collaboration can be supported in many different ways. For some areas of development, basic collaboration support provided by concurrent version systems may be feasible. In other areas, such as process creation support, the ability to synchronously work on a process model may be desirable. In this case, there are domain experts discussing an overall application flow, possibly even consulting Grid middleware experts.

Integration of Management and Development Components

The separation of the different roles of users that was introduced in chapter 1 is much stronger in the case of a classical Grid system than for the ad hoc Grid. A classical Grid system combines large resources managed by dedicated administrative personnel in an infrastructure that support the execution of distributed applications that are developed by Grid experts and used by Grid users that typically do not hold the necessary know-how in Grid software development or distributed application development in general. An ad hoc Grid, on the other hand, can combine dedicated large resources with, for example, personal computers where resource owners join a collaborative network and share their assets in return for the ability to solve their own problems as application users. Every one of those users assumes at least the role of a Grid user and an administrator. An ad hoc Grid environment needs easy to use management components that integrate well with the automatic management functionality provided by the runtime system. Additionally, certain management functions are important also to the development components. Consequently, the management components should be integrated with the development components.

2.4 Summary

In this chapter, an overview of three different application scenarios for a service-oriented ad hoc Grid application was given. The three applications come from the medical and mechanical engineering application area as well as from media research. These distinct application areas all present use cases where a service-oriented ad hoc Grid could improve the situation of individuals with a high demand for collaborative use of distributed resources.

After having presented the sample application scenarios, a number of requirements for an ad hoc Grid middleware solution were presented. These requirements are mainly derived mainly from the sample applications introduced in the first part of the chapter. Additional requirements for a service-oriented Grid may exist, yet they are not focused in the scope of this thesis.

3

Related Work

3.1 Introduction

In this chapter, related work in the field of service-oriented ad hoc Grid computing is discussed. The discussion is roughly split into three sections. First, an overview is given of overall projects aiming at creating a more dynamic Grid computing environment. Many of those projects use proprietary technology that is not standards compliant with other service-oriented Grid solutions (if they are service-oriented at all). In the second part, the related work for a number of components of a service-oriented ad hoc Grid middleware is presented. After this discussion of the different aspects of the runtime environment, related work for Grid development and management components is presented.

3.2 Overview

There are several projects investigating a more dynamic form of Grid computing than the standard Grid solutions currently offer. These dynamic Grids are featured under many names, e.g. ad hoc Grids, personal Grids, desktop Grids, P2P Grids, dynamic Grids. In the following, we discuss related Grid projects which focus on a dynamic Grid environment and attempt to automate the configuration and maintenance of such a Grid to enable easy adaptation and use.

Sterck et al. [160] introduce a lightweight Grid environment for solving bioinformatics problems on small, "privately operated" Grids. An explicit design choice was made not to use standard Grid middleware solutions like Globus, justified by the reasons that they are too cumbersome and difficult to use in a dynamic environment and are not flexible enough to facilitate the e-science re-

searchers' needs. Basic design criteria of the system are decentralization, provided by an underlying tuple space concept, and platform independence, provided by an implementation in Java. The following design issues are dealt with within the framework: scalability, resource allocation and scheduling, automatic distribution of application code to workers, inter-process communication and resource discovery. Furthermore, the following design issues were presented but are not yet implemented: secure resource sharing, a user billing system and quality of service mechanisms. All inter-process communication is done via Java sockets and serialized Java objects. This makes the configuration of the framework difficult to manage once organizational boundaries need to be crossed and restricts the scalability of the framework. Although this is not an issue for the scenario described in the paper, it limits the usability of the framework for other research projects. The fact that standard toolkits and protocols were avoided for the sake of usability makes it difficult to utilize components from this project or integrate new developments from other projects.

A desktop Grid computing environment for enterprise solutions is presented by Naik et al. [125]. Each Grid node runs a VMware Workstation instance with Linux as the guest operating system (OS). On each guest OS, IBM's WebSphere Application Server AEs 4.0 is run to host web services which are integrated into the Grid applications. The system is targeted at enterprises which install and manage the virtual PCs and the application servers to create a small but secure Grid environment. Ease of use and manageability are not dealt with since they are not seen as critical issues.

Shivle et al. [149] present resource allocation algorithms which can deal with node failures in a dynamic Grid environment. The Grid in this work consists of mobile nodes, and the application is modeled as a directed acyclic graph. Shivle et al. do only address algorithmic issues, the underlying middleware to their application scenario is not discussed.

Glatard et al. [92] and Germain et al. [91] both discuss work in the field of medical Grid computing. The work presented by Glatard et al. introduce a workflow engine for medical applications with high data throughput, and Germaine et al. introduce a Grid based image analysis approach. The authors state that the integration of workstation PCs into a desktop Grid is a growing interest in the medical community. Unlike the projects mentioned above, the solutions presented in these papers are not based on lightweight custom developed Grids, but on a standard service-oriented Grid middleware. The papers state that implementing the medical applications on the standard Grid middleware is far from trivial due to the complex nature of both the application and the middleware itself.

Andrade et al. [13] introduce a peer-to-peer based middleware which operates on a Bag of Tasks (BoT) in a dynamic environment where nodes can leave and join the Grid at any time. A BoT is the framework's unit of scheduling and has the following characteristics: (i) it does not need any synchronization between tasks, (ii) it has no dependencies between tasks and (iii) it can tolerate faults

caused by resource unavailability with very simple strategies [14]. This places many restrictions on the application developer. The peer-to-peer system and the BoT have their own programming API and do not make use of standards.

Several criteria for ad hoc Grid computing are discussed by Amin et al. [8], and the need for peer-to-peer integration is explained. A figure illustrates the combination of the JXTA [139] peer-to-peer infrastructure and the CoG [175] framework. The details of the realization of the concepts are not shown, however. In a follow-up paper [9], the authors describe an architecture for an ad hoc Grid which focuses on the issues of community control, quality assurances, and spontaneity of service contribution and invocation, developed as an integral part of the Java CoG Kit. The security issues of identity management, identity verification, and authorization control are discussed in another work by Amin et al.[10]. The authors see security related issues, especially providing access control in a spontaneous Grid collaboration as the main issue that needs attention by researchers [7].

Chakravarti et al. [39] introduce an organically inspired peer-to-peer model to facilitate the use of desktop Grids. The organic Grid is meant to bridge the gap between traditional Grids and centrally managed distributed computation projects like Seti@Home. The paper concentrates on the autonomic scheduling of simple independent tasks and does not describe the Grid architecture in which the applications run.

Huang et al. [98] discuss different scenarios for on demand service provisioning in Grid environments. They distinguish several models for such service scenarios based on homogeneous platforms, virtual machines, or on demand implementation based on interface descriptions. However, their implementation description does not approach many of the aspects found in real world Grid scenarios.

Bertino et al. [28] propose an approach to supporting fine-grained access control for Grid resources. The authors argue that such a fine-grained policy-based access control is necessary to enable desktop Grid computing. Giving resource owners a higher flexibility in controlling access to their resources is seen as a vital requirement for the adoption of the Grid paradigm to a higher extent into new avenues such as desktop Grids. Similarly, in the proposal, Grid users should get a higher flexibility in choosing the resources in which their jobs must execute. The actual Grid architecture is not described.

A lightweight Personal Grid based on a super-node peer-to-peer network is introduced by Han and Park [95]. A hierarchical scheduling system is used to distribute different types of applications in the network. The programs are described as services and are advertised by publishing XML descriptions through the peer-to-peer framework, but service-oriented standards like WSDL or SOAP are not used. Great emphasis is placed on creating a light-weight and easy to use system in contrast to the complex Grid toolkits like Globus or Unicore.

The goal of ASKALON [62, 64] is to simplify the development and execution of workflow applications on the Grid. It contains a number of high level

services including a workflow scheduler and execution engine, a resource manager for computers and application components and a performance prediction service. ASKALON workflow development is based on performance monitoring and prediction services that form the basis for the workflow development support provided by the system. Workflow specifications are based on the proprietary Abstract Grid Workflow Language (AGWL) [63, 65], they lack support or foundation on an accepted standard for specification of Grid processes.

The Vienna Grid Environment [26, 30] promises on-demand supercomputing. Work in the context of VGE focuses on quality of service aspects in service-oriented Grid environments, many aspects of ad hoc Grid computing such as service deployment and intra node security issues are not addressed.

3.3 Ad Hoc Grid Middleware Components and Solutions

3.3.1 P2P Discovery and Communication

Iamnitchi and Foster proposed a P2P approach to resource location on the Grid, based on a message forwarding and flooding algorithm [99, 100]. Their first approach was not very scalable and not designed for a service-oriented Grid environment. Later [69] they argue towards more research on the confluence of P2P and Grid computing.

Talia and Trunfio [168] also argue towards further evaluation of the possible synergy effects between P2P and Grid computing. In [167], they introduce an extension to the original Gnutella protocol called GridNut that is intended as a discovery mechanism for OGSI Grid systems. GridNut, like many other P2P and Grid projects, is only intended to support resource discovery, no provisions for efficient communication through an underlying P2P network are provided.

In addition to the early flooding based discovery approaches, different groups have studied Grid service discovery on the basis of various structured P2P overlay networks.

Cheema et al. [43] propose a resource discovery scheme based on the Pastry P2P framework. Static as well as dynamic properties of the nodes in the Grid environment are directly encoded in the 160-Bit addresses used by Pastry. Search requests are generated by using the same algorithm to encode the exact specification in the 160-Bit address vector. A fundamental problem of this approach is its limited property capacity of only 160-Bit and its lack of support for range queries that require generation of multiple concurrent requests for all imaginable values and resulting addresses. This approach would require 2^{160} queries to discover all nodes of a Grid.

Schmidt and Parashar [147] propose an approach to web service discovery in P2P distributed hash tables based on space filling curves that allows for range

queries about keywords. The work does not explicitly state which parts of a web service description should be selected as keywords. A similar approach has also been proposed by Andrzejak and Xu [17] and Chen et al. [44]. While those approaches address the problem of supporting range queries in distributed hash tables, they still lack expressive power for queries considering the values of several feature dimensions of a record.

Xia et al. [179, 180] present a decentralized Grid service discovery system based on a structured overlay network. Their approach spans a custom overlay network, taking virtual organization structures into account, with the use of a gateway discovery server per virtual organization that are combined in a structured overlay network. A similar approach combining discovery servers and structured P2P overlay networks in a hybrid topology is also addressed by Jie et al. [102] and Gong et al. [94]. A number of resource discovery proposals for Grid environments has emerged that build on the JXTA P2P infrastructure [12, 150, 107]. Due to the nature of the JXTA network, the resulting discovery infrastructures use a super-peer network to implement search queries very similar to the previously introduced approaches using hybrid client/server networks.

Neither of the aforementioned approaches considers the communication capabilities as an asset for Grid service communication and invocation similar to the approach presented in this work. All of the approaches focus on the algorithmic capabilities of their solutions and rarely discuss the integration of their work into a standards-compliant, service-oriented Grid middleware.

Banerjee et al. [22] address scalability issues of standard Web service registries following the Universal Description, Discovery and Integration (UDDI) standard by connecting several local UDDI registries to a DHT index. They propose to register every possible search term from the local UDDI registry in the DHT to enable a two phase discovery first performing lookup in the distributed index to identify the local registries that can give detailed answers for the service descriptions. Even though, this approach uses the UDDI standard, it can hardly incorporate dynamic properties of a Grid environment. Such dynamic properties are often included in the resource information managed by standard Grid indexing services such as the Monitoring and Discovery Service (MDS) of the Globus Toolkit.

In addition to the previously mentioned works regarding the discovery of Grid services based on certain technical or syntactical features of the service description, a number of papers have been published addressing a more semantically rich discovery mechanism for resources in the service-oriented Grid. In this area, Cao et al. [35] describe an ontology based approach to resource description and discovery. Heine et al. [96] discuss the possibility to use topic maps and description logic concepts to construct a resource discovery overlay network. Similar to the latter approach, Li and Vuong [114] describe a query routing approach based on an P2P overlay network using RDF descriptions for Grid resources.

Few groups report on using a P2P framework and the general communication

capabilities exposed by such a framework to realize service communication across network barriers in a service-oriented Grid context. Caromel et al. [36] describe the use of a web service dispatcher that acts as an intermediary component for web service invocation across firewalls. The approach uses an explicitly configured and used SOAP message dispatcher and does not use the firewall transition capabilities of a general P2P communication framework such as the system presented in this thesis. Genaud and Ratanapoka [90] describe a proprietary system providing a P2P based message passing architecture similar to the functionality of the Message Passing Interface (MPI). However, their work does not consider passing of SOAP messages for service invocation or alignment with standard compliant service-oriented middleware.

Fox et al. [73, 136, 72] propose the NaradaBrokering middleware as a communication infrastructure for Grid environments. NaradaBrokering is a broker architecture built upon the JXTA P2P framework. Message brokers are realized as group services in the network, interconnecting several groups in a super peer network for message exchange. The focus of the Narada system is on message brokering and protocol translation. Therefore, SOAP messages are transformed into a proprietary protocol and transmitted through the JXTA network. Narada also contains SOAP inspection and processing components to alter destination addresses in transit. While Narada realizes a message oriented middleware that can only forward messages, it lacks a distributed hash table that can be used to store additional information records.

Ganguly et al. [87, 88] propose a different heavyweight approach to enable Grid communication over firewalls and network address translation (NAT) routers. Their approach is to use virtualization of an entire host containing a Grid node and the construction of an IP network connecting the virtual machines over an underlying P2P infrastructure. All network traffic between the virtual machines is intercepted in the virtualisation layer and tunneled between the virtual machines.

3.3.2 Hot Service Deployment

The Globus Toolkit 4 (GT4) offers mechanisms to create service instances on a Grid node via service factories. The GT4 does not offer a flexible mechanism for hot factory deployment (i.e. the deployment and removal of a Grid service without restarting the OGSA web application inside the web application container). Instead, a service is deployed using two subsequent Apache Ant [169] tasks. The files making up a Grid service are copied into the OGSA directory structure before the entire OGSA application is copied into the application container. To effect the changes, the application container must restart the OGSA application, causing a restart of every service running in its context. Since the first publication of the Hot Deployment Approach presented in this thesis [81], the Globus Toolkit developers are actively pursuing a similar approach that will

be included in future versions of GT4 [141]. In [174] the authors motivate the need for application, component and service deployment for Grid environments. However, the focus of the work is on supporting the creation and maintenance of software packages that can easily be installed by an administrator, it lacks support for non-disruptive deployment of services.

The architecture of MS.NETGrid [32] and OGSI:NET [176] are based on the GT3 core and use the Microsoft Internet Information Server (IIS) as their hosting container. Dynamic deployment of services is not addressed by MS.NETGrid or OGSI:NET, and the deployment process is similar to a Tomcat based GT4 distribution. It requires the alteration of the container's configuration file, copying of service descriptions as well as service assemblies (service DLLs) and the restart of the IIS.

OGSI::Lite [135] attempts to provide an OGSI implementation in the Perl language. Every OGSI::Lite service is run as a Perl CGI module inside Apache and can therefore be dynamically replaced by replacing the file on the server. The usage of Perl, however, creates some deficiencies, such as hard to understand and reuse code, only rudimentary support for WSDL creation - the authors suggest to write Java interfaces and generate WSDL definitions for the Perl services from them - lack of feature implementations in SOAP::Lite and the missing support of the multitude of Java web service developments (e.g. web service security).

The PyOGSI [111] project aims to make the Globus Toolkit accessible through a Python interface. It does not address the problem of service deployment.

The Imperial College e-Science Networked Infrastructure (ICENI) [86] project is aimed at creating a service-oriented architecture for Grid computing. It was originally [85] built on Jini technology [163] with Java RMI as its communication protocol. Recognizing the impact of XML-based protocols, ICENI was recently ported to JXTA and OGSI. The OGSI implementation of ICENI only uses GT3 as a communications provider rather than extending the functionality of the OGSI implementation. ICENI services are exposed as OGSI Grid services via a complex process using introspection and the generation of proxy classes. The need for a deployment service for preparation and instantiation of a service associated to a hardware resource is expressed in a recent publication outlining future plans for the ICENI system [119]. However, the deployment functionality sketched in this plan considers deployment on a coarse grained scale adopting entire operating system virtualization, it lacks a lightweight solution for non-intrusive deployment of single services.

GLARE [151] is described as a Grid activity registration, deployment and provisioning framework built on top of GT4 with a focus on shielding the details of physical resource management from application developers. Instead, the framework should support easy and efficient execution of higher level Grid workflows. The developers of GLARE describe an on-demand deployment service that executes Ant or GNU automake scripts to deploy user software on demand on the target Grid node. While this approach deals with the installation of complex

applications, the approach only exposes an installation mechanism similar to the standard deployment mechanism provided by the underlying Grid middleware. It cannot install new Grid services in the Grid service container without requiring a restart of the container. Also, the appropriate rights to execute the installation scripts are required for the target Grid node, issues of service isolation are not addressed.

Furthermore, component deployment is an active research issue in a more general context than service-oriented Grid computing [59]. The most relevant work for web service deployment is done in the field of web service frameworks and web application containers. For example, the AXIS web service framework [19] provides a manager application to add and remove web service definitions during runtime. However, the developer is required to make sure that the needed classes are available to the AXIS web application. The installation instruction explicitly states the need to reload the entire web application, effectively shutting down any other service currently running [19, Step 5] under the engine's control. It mentions the need for the enclosing web application container to dynamically reload applications during container runtime, a feature offered by Apache Tomcat [20].

JBoss [66] offers hot deployment for various deployment units (including plain JAR files, web application archives etc.). Using this feature to allow hot Grid service deployment on top of GT4 would nevertheless require the dynamic update of GT4's internal Grid service registry. Instead of relying on the JBoss hot deployment mechanism, the current implementation of the hot deployment service introduced later in this thesis remains independent of the application container.

3.3.3 Security

Security is a major aspect in an ad hoc Grid since there is always the possibility that a node introduces malicious code. On demand computing with dynamic service deployment creates several new security aspects that must be dealt with beyond the standard security requirements existing in previous Grid systems. Therefore, the focus of the security considerations in this work lies on the isolation of Java Grid services that share a common Grid service container. Aspects of isolation include protection of the Grid service instances against each other and protection of the underlying resources against the service instances.

Emerging proposals for isolation of different Java threads in the same Java Virtual Machine (JVM) address security threats arising from the sharing of a single JVM between different applications [165]. An approach to implement such a shared JVM is actively pursued by Sun Microsystems in the form of the multitasking virtual machine (MVM) [48]. While MVM offers mapping of isolated Java threads to operating system processes, only a prototypical implementation for Sparc Solaris has recently been published and would require the availability of the MVM for the hosting system and a switch to this new JVM. Porting

a service hosting environment like the Globus Toolkit and Tomcat to the new virtual machine VM requires substantial efforts to change those systems [103]. The MVM implementation lacks being easily portable to other JVM implementations, a portable add-on solution for native components of Grid services that is interoperable with different virtual machines would be beneficial.

The possiblity to decouple native processes from the process space of the Java virtual machine has been investigated by Czajkowski et al. [49] in order to achieve better robustness of Java applications relying on native methods, not to enhance security of a shared Java environment or the underlying operating system. Systems like the Entropia Virtual Machine for Desktop Grids [34] or GridBox [52] propose the application of virtualization and sandboxing technologies to achieve security for native Grid applications. They can only isolate entire native applications. Applied to the scenario of Java Grid services relying on native components, they cannot provide isolation of different services inside a single JVM.

3.3.4 Data Handling

There are several approaches leading to more flexibility and better performance for binary data transmission compared to standard SOAP. Allcock et al. [5, 6] introduce GridFTP as a high performance data transfer protocol. GridFTP starts an extra process to efficiently transfer data from one node to another. It is completely decoupled from the service and thus violates service-orientation. Furthermore, GridFTP is not very flexible, since data cannot be dynamically transferred after service start. RFT [170] is a front-end Grid service which controls GridFTP.

Abu-Ghazaleh et al. [2], [1] introduce differential serialization/deserialization to improve the performance of serializing and deserializing SOAP messages. Since transferring large amounts of data over SOAP is avoided altogether by the approach presented in this work, the performance loss resulting from the serialization and deserialization of large amounts of data is avoided.

Seshasayee et al. [148] introduce SOAP-bin and SOAP-binQ as a combination of SOAP with an efficient binary protocol. Although they are alternatives to SOAP with Attachments (SwA), they have the same disadvantages. The order of attachment transmission cannot be chosen and overlapping of service execution and data transmission is not possible.

An alternative to MIME named Direct Internet Message Encapsulation (DIME) is presented by Nielsen et al. [129]. DIME enables random access to each message part of a multipart message. A performance comparison of binary data transmission using SOAP and using DIME is presented by Xue et al. [181]. Ying et al. [182] measure the performance of SwA. Though SwA is significantly faster for large amounts of data than SOAP, it lacks flexible in transmission patterns for large amounts of data. A similar lack of flexibility can be found in all the previously presented proposals, they do not support dynamic transfer of data during data production or service execution.

NaradaBrokering is an open-source messaging infrastructure based on a network of message brokers. In NaradaBrokering events are used to encapsulate different information, i.e. NaradaBrokering is an event brokering system. Fox et al. [72] present an approach to interface NaradaBrokering with web services. Since NaradaBrokering is based on brokers collaborating in a broker network, it differs significantly from the ad hoc Grid architecture discussed in this work and especially from the data handling approach presented in this thesis.

3.3.5 Process Execution Support

Yu and Buyya [183, 184] present a taxonomy of Workflow [1] specification, management and execution systems for the Grid . Most of the Grid workflow projects discussed by Yu and Buyya implement their own graphical workflow specification environment and graph based workflow language, leading to replication of work due to the lack of standardized syntax and semantics for Grid process description languages. Consequently, the authors argue in favor of further work towards standardization of Grid process modeling and specification languages. Yu and Buyya identify some key issues requiring attention for the successful application of Workflow execution techniques in production Grid environments but are lacking in current workflow execution approaches. These issues include modeling and handling of QoS aspects, trust and market mechanisms for scheduling and the implementation of strong fault handling mechanisms.

The argument towards establishment or application of existing or common workflow execution standards to Grid computing is also presented by Leymann [113]. The author argues in favor of adopting the Business Process Execution Language for Web Services (BPEL) standard [133], discussing the possibility of integrating BPEL with the WSRF set of specifications for interaction with Grid services and management of Grid processes. Leymann acknowledges the possibility of extensions to the BPEL standard to reflect requirements of the service-oriented Grid similar to other extensions already made for other application areas.

Applicability of the BPEL standard to the service-oriented Grid environment is analyzed by Chao et al. [41], Akram et al. [4] and Emmerich et al. [58]. While the first group comes to the conclusion that BPEL can be used to orchestrate Grid services based on GT3, they see considerable problems regarding BPEL support for the Grid service standards and support for e.g. WS-Addressing in the BPEL engine. These limitations are dealt with in the Grid process execution engine described in this thesis. Both other groups identify a number of requirements for a Grid service based process specification and enactment system and compare the requirements to the capabilities offered by the BPEL standard. Both groups confirm the general applicability of the BPEL standard in a scientific application

[1] The terms *Grid workflow* and Grid process are used synonymously throughout this work.

environment on the service-oriented Grid. While Akram et al. evaluate BPEL on a conceptual level, Emmerich et al. evaluate the BPEL standard based on experiences gained by implementing a sample application based on a modified ActiveBPEL engine. They clearly identify problems in the parallel and concurrent execution of parts of a scientific workflow that are answered by using additional logic in the process specification. A downside of the approach by Emmerich et al. is the increased complexity of the application logic by expressing certain constructs with additional logic instead of providing such constructs with a strong semantic in process specification language and the corresponding runtime system.

Amnuaykanjanasin and Nupairoj [11] address the issue of integrating the standard Grid service infrastructure (implemented for the Globus Toolkit 3) with a BPEL execution engine. They show a sample application for drug design relying on the BPEL engine as the core workflow control component of the application. They do, however, not directly invoke Grid services, generated hard-coded web services are used as proxy services to invoke Grid services. The BPEL engine uses the same proxy web service that dynamically switches the Grid service endpoints, losing detailed information about the partner Grid services in the process specification.

3.4 Grid Development Environment

Tools supporting the development of service-oriented Grid applications have not seen much attention since most groups working in the field of service-oriented Grid middleware focus on the runtime components of the Grid middleware. Most effort can be seen in the field of specification support for the predominantly proprietary languages for Grid process specifications.

3.4.1 Grid Service Creation

Since service-oriented Grid computing is an extension of the service-oriented computing paradigm, there are several relevant papers dealing with applying model driven development principles in a web service environment.

A WSDL centric approach is presented by Mulye [123]. The paper discusses which components of a service belong to which layer of the Model Driven Architecture (MDA) approach. Definitions, Operations, Port types, Messages, Parts and Part types are placed in the platform independent model (PIM) layer. Service, Ports and Bindings are placed in the platform specific model (PSM) layer. This mapping is controversial since most of the components placed in the PIM are specific to a service-oriented approach and should therefore be placed in the PSM layer. The paper goes on to suggest that a document centric view is better suited to service-oriented models than a Universal Modeling Language (UML) centric view.

A WSDL free approach is presented by Grønmo et al. [130]. A WSDL independent UML model is proposed because it offers a much clearer view of the system functionality. Automatic transformations from UML to WSDL are used to create the actual WSDL document. Since the UML model is completely free of any WSDL specific components, the developer is free to concentrate on actual business concerns. The downside is that an integral part of service-oriented systems is no longer visible in the MDA models and thus outside of the development scope.

Gokhale et al. [93] describe a Model Integrated Computing Tool called CoSMIC which deals with resource reservation and component deployment for their Grid middleware Grit. However, Grit is a CORBA based middleware which was not designed for service-oriented Grid use. Furthermore, their work does not deal with the transformation of PIM models to PSM models or separation of concerns.

Similarly, there are works that do not address the problem of transforming PIM to a Service Oriented Architecture (SOA) specific PSM or the separation of concerns in the PSM or code layer while dealing with service-oriented architectures and MDA in general, e.g. Rakesh et al. [142, 74, 152]. Andreozzi et al. [16] use measurement theory and the Logic Scoring of Preferences method to select Grid services. The authors describe a formal model for satisfaction based service evaluation which can be used in MDA based Grid computing. The paper does also not deal with the transformation of PIM models to PSM models or seperation of concerns.

Manset et al. [116] combine a formalized architectural description with the model-driven engineering process proposed by the MDA. Great emphasis is placed on the formal specification and verification of a large number of sub-models using a special architecture description language. The complexity of the formal specification and the large number of sub-models make this a hard to adopt approach not suitable to application domain experts.

Morohoshi and Huang present a toolkit to ease Grid service Development for the Globus Toolkit 3 (GT3) [121]. This approach combines custom wizards to specify the properties of the service under development with standard generator tools for e.g. WSDL generation in a custom GUI. The toolset described by Morohoshi and Huang does neither employ a model driven development approach nor does it support round-trip engineering of Grid services. Furthermore, it lacks integration with a strong development environment such as the Eclipse platform that bars it from using and extending all additional functionality provided in the Eclipse ecosystem.

Mizuta and Huang present another approach to GT3 service generation using a model driven approach based on the AndroMDA toolkit [120]. While this second proposal employs a model driven approach, it is still geared towards generation of a Grid service, not round trip engineering. Furthermore, the approach is geared towards a single Grid platform and lacks the inherent design approach to allow fast adaptation to other service-oriented Grid middleware target systems with

reuse of significant parts of the implementation of the toolset.

The Globus Services Build Tools (GSBT) project [159] realizes a number of Ant scripts that help build a service based on the code generation tools included in Axis and the standard Globus Toolkit release. An Eclipse plugin has been developed as part of the GSBT that collects a number of wizards to create initial input for the command line oriented scripts but does not support round trip engineering or higher level development functions such as support for Grid process development and debugging.

3.4.2 Grid Process Development

Recently, the development of a standard compliant BPEL editor for the Eclipse platform has begun [53]. The large and active developer community of the Eclipse project and the commitment of industrial partners like IBM and Oracle to the project make it an interesting development. The BPEL editor is a general purpose BPEL editor that does not explicitly address Grid process creation or synchronous collaboration. Furthermore, it does not distinguish between expert users and non-experts in the presentation of the process under development and available tools.

Another effort to develop an Eclipse based process editor for a Grid environment is undertaken by the OMII-BPEL project [58]. Similar to the Eclipse BPEL editor, this effort lacks support for synchronous collaboration. It is aimed at developing processes specifications for the OMII-BPEL execution engine, which is an extension to the ActiveBPEL engine.

3.5 Summary

In this chapter, related work in the field of service-oriented ad hoc Grid computing was discussed. The discussion of related work starts with an overview of different projects aiming at realizing a more usable, flexible and dynamic Grid environment often described as personal Grid, desktop Grid, P2P Grid, dynamic Grid or ad hoc Grid. After this project overview, different components and aspects of the service-oriented ad hoc Grid middleware presented in the following chapters were addressed in the presentation of related work in the respective field. This discussion of related work for the various components of the ad hoc Grid middleware presented in this thesis was split into two parts focusing on runtime infrastructure and development support.

To summarize the work conducted in the overall ad hoc Grid related projects, the related work can be divided into two categories. The first deals with application specific research and the second deals with frameworks which aim to create a generic Grid environment. Developers focusing on an application often prefer a very lightweight and easy to use Grid environment [91, 92, 160, 149] and

actively avoid heavyweight middleware or draw attention to the development and administrative overhead induced by the Grid middleware. The downside of this approach is that basic functionality like accounting, billing and security are only dealt with in a rudimentary fashion, and the solutions barely scale beyond the current application's scope. On the other hand, the generic Grid environment developers try to offer a feature rich environment to host many different kinds of applications at the cost of creating a steep learning curve and high administrative costs. Due to time and resource constraints, many generic middleware projects still have a specific focus of excellence [9, 39, 28, 13, 95, 26]. While all of the projects provide insightful research, it is very difficult to combine the research results in a single environment, due to the fact that many of these projects create their middleware from scratch. In the multi-disciplinary, multi-organizational and multi-application field of ad hoc or desktop Grids, only a cleanly designed, modular and standards based approach is able to fulfil the application developers' needs while at the same time offering the extended functionality beyond the applications' scope. The service-oriented computing paradigm offers the basis on which such a modular approach can be realized.

For the work specifically focusing on the different components of a service-oriented ad hoc Grid middleware, a lot of effort has been put into the realization of service and node discovery mechanisms based on the P2P paradigm. In the other middleware component areas enabling a true service-oriented ad hoc Grid environment, many deficiencies exist. These deficiencies are addressed in the following chapters of this thesis.

4

Design of a Service-Oriented Ad Hoc Grid Infrastructure

4.1 Introduction

This chapter introduces the design of a service-oriented ad hoc Grid middleware solution. The chapter's intention is to give an overview of the system and motivate the design decisions for the most important components. An actual implementation of an ad hoc Grid solution based on this design - The *Marburg ad hoc Grid Environment* (MAGE) is described in chapter 5.

The MAGE system is designed as a combination of a runtime execution environment - the actual Grid middleware supporting service-oriented Grid applications - and a management and development environment that supports application developers, application providers and end users in setting up and maintaining their ad hoc Grid environment.

For many components in the design, a number of options for a concrete design are proposed and discussed. The actual decision for the final system design often relies on the functionality of a concrete third party component chosen for the implementation of the system. Also, the design of the ad hoc Grid middleware runtime as well as the development and management environment are presented with respect to the most important design decisions influenced by the *ad hoc* nature of the middleware under development. Many common design aspects directly inherited from a service-oriented Grid middleware that can be used as a basis for a concrete implementation are not discussed in this chapter since they exceed the scope of this work.

The rest of this chapter is separated into two parts. In the first part, the design of the runtime environment for an ad hoc Grid middleware is presented,

in the second part the design of an integrated development environment for the ad hoc Grid is discussed. The discussion of the runtime component introduces the fundamental choice of technologies for the underlying P2P infrastructure on which the design for the node and service discovery component as well as the P2P communication are founded. The design of the components allowing hot deployment of services and ensuring security of the Grid platform as well as service instances from different providers and users of the Grid are introduced after these initial P2P related components. Higher level runtime components for flexible handling of data transmissions and the efficient execution of complex Grid processes conclude the first part of this chapter. Several aspects of the ad hoc Grid middleware runtime component design and the motivation for the design decisions have been published in [77, 81, 80, 78, 79, 97, 118, 83, 156, 155, 153, 154]. The second part of this chapter gives a short introduction to the application of model driven development to service-oriented Grid environments. After this introduction, the design of the components of supporting Grid service and Grid process creation in an integrated development environment are discussed. The second part of the chapter is then concluded by a discussion of the debugging and Grid management components designed for the integrated development environment for the ad hoc Grid. The work presented in the second part of this chapter has partially been published in [157, 82, 158].

4.2 Runtime Environment

A first goal in the design of an ad hoc Grid middleware must be the runtime component that will form the basis for ad hoc Grid applications. Figure 4.1 shows an overview of the components included in the runtime system of the ad hoc Grid environment reported on in this thesis. At its basis, the middleware uses an existing P2P infrastructure that provides discovery functionality for Grid nodes and services as well as message or connection oriented communication capabilities to the ad hoc Grid environment. The P2P discovery capabilities are exposed to users and applications through a Grid service interface similar to other key capabilities of the platform for process and service management. Hot deployment of services acts on secure isolation contexts for services that include isolation of legacy applications. Access control to services is provided by standard mechanisms from a basic service-oriented Grid middleware (GT4 in case of the MAGE implementation). The data handling mechanism included in the ad hoc Grid environment enables flexible transmission of large amounts of data with high performance in a web service-oriented environment and forms a basis for many Grid services as well as for the execution of higher level Grid processes in a process execution engine. The Grid service container and process execution engine share the same web application container while residing in different application contexts.

The different components of the ad hoc Grid middleware have been designed

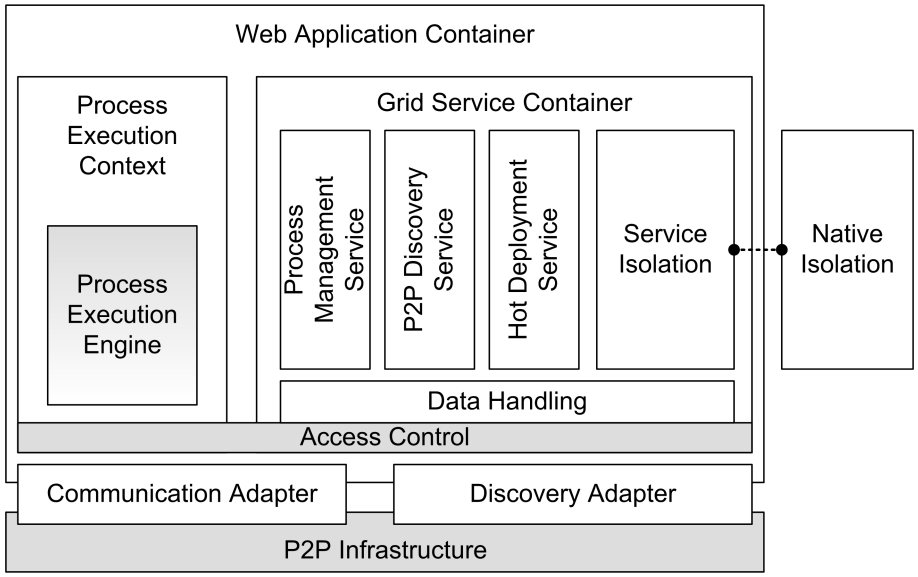


Figure 4.1: Overview of the components of the ad hoc Grid middleware.

using a modular approach with highest possible independence. Many of these components such as data handling and P2P communication for Grid service interaction are also applicable in other web service environments.

4.2.1 Peer-to-Peer Infrastructure

A suitable peer-to-peer (P2P) infrastructure is beneficial for providing solutions to the problems of barrier-free node communication and for node/service discovery. Since a P2P infrastructure will be used as a fundamental component in the ad hoc Grid infrastructure discussed in this work, a presentation of the properties of different P2P infrastructures is given in this section.

P2P systems [139, 145] are typically constructed as overlay networks to the internet infrastructure, forming a uniform virtual address and routing space over different physical networks. Their purpose is the collection of various resources in a searchable information space. One possible classification of P2P systems distinguishes between systems that use some form of flooding for search (e.g. [177]) and systems that construct an index structure for meta-information (e.g. [161]).

Flooding describes the propagation of search requests throughout the network, assuming that every peer receiving a request acts as a relay and forwards the message to neighboring nodes. Results matching the search request are then transmitted to the initiator of the request either through a direct connection or again by multi-hop propagation through the network. Different optimizations have been proposed, since a naive implementation of a flooding mechanism for search requests leads to scalability problems. Those optimizations include the

construction of optimized any-cast routing trees and super-peer structures. Any-cast trees are overlay routing structures that allow one-to-many broadcasts of messages from every node in the network. Super-peers are well connected nodes with a large amount of available network bandwidth that collect meta-information from many so called "edge nodes" with only limited bandwidth and connectivity. Search requests are only propagated between and answered by the super-peers, relieving the weak edge nodes from the burden of propagating search requests, so they can use their network connection for actual work.

P2P networks using flooding for search requests share two key properties. As a positive feature they provide very strong search semantics; the request is seen by every node on the network, therefore, the search can consider the entire content of the resource or associated metadata. Queries can easily use multiple wildcard patterns as long as the local search engine for resources supports them. A down-side of the flooding search mechanism is the outreach of the request. In order to limit the bandwidth used for propagation of requests and results, the number of times a packet is forwarded is typically limited. This results in a so called "horizon" for every peer. There is no guarantee that a search request reaches every node in the P2P network, peers and their resources behind the horizon are invisible to a peer. This implies that answers to a search request are neither complete if results are returned nor definitive if no result is returned. There is no guarantee that *all* resources matching the request are reported and resources actually matching the request might exist behind the horizon of the requestor. Stronger guarantees on the semantics of search operations are provided by the second class of P2P systems. These systems construct a consistent information space out of the combined resources provided by all the nodes in the network. This information space has the features of a distributed hash table (DHT) and may be used to store and retrieve key-value-pairs. Other than in a flooding network, an answer from a DHT is definitive. To a high degree of confidence, all information items matching a query in the information space are returned by the system. If no item is returned it can be assumed that no item exists in the information space that matches the query.

4.2.2 Node and Service Discovery

An ad hoc Grid middleware solution must address the requirement for node and service discovery as introduced in section 2.3.1. For the expected high churn rate (i.e. the high fluctuation of nodes in the Grid environment), discovery mechanisms based on a P2P infrastructure present themselves as good solutions, since P2P networks and routing algorithms are specifically designed to cope with high churn rates.

Every node of an ad hoc Grid system has a number of static properties such as information about the hardware and the operating system and pre-installed applications of the node. Additionally, it has a number of dynamic properties

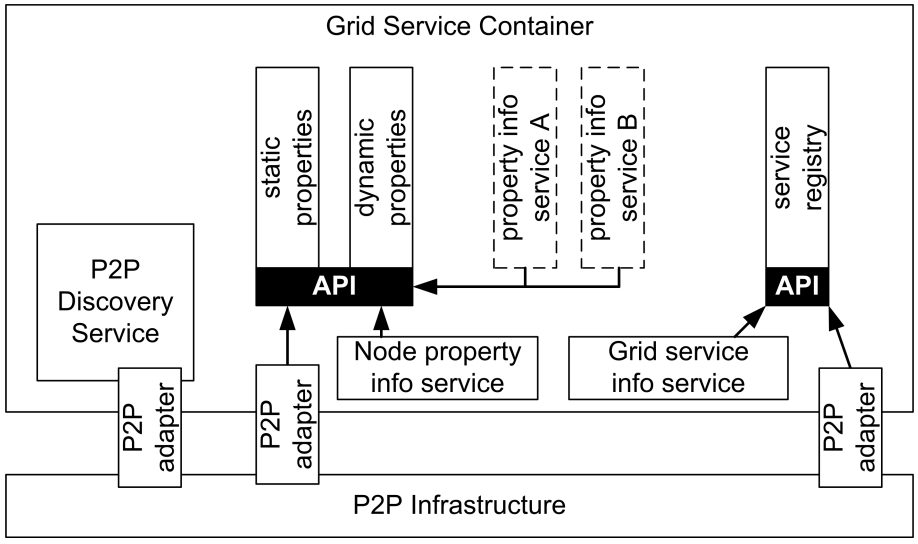


Figure 4.2: Conceptual overview for the integration of a P2P infrastructure with the ad hoc Grid middleware.

such as the current utilization of the different hardware resources. There are two typical use cases for this node information in the ad hoc Grid:

- Selection of a node for the deployment of a Grid Service based on the static properties of the node.
- Creation of a WS-Resource instance or selection of a node for the deployment of a Grid Service based on the dynamic properties of the node.

In order to abstract the implementation of the P2P network from the core information service, any implementation of the ad hoc Grid infrastructure must provide a local interface to this node information record. A P2P connector to the repository may then either actively publish the information into a DHT information space or provide a connector for queries received over the network. This architecture is shown in figure 4.2. In addition to non-directed discovery queries transmitted over the P2P network, an information service should provide access to the node information record via a standard Grid service interface.

The node discovery mechanism must support arbitrarily complex queries over both the static as well as the dynamic properties of a node. While most current DHT implementations do not provide strong enough search semantics, pure flooding networks bear an inherent search horizon problem. Application layer multicast trees on top of structured P2P networks on the other hand provide certain guarantees for the reliable transmission of a message to every node in the tree. Furthermore, many implementations claim good scalability of such a solution [38]. The number of nodes in an ad hoc Grid group can be assumed to be reasonably small for the use of an application layer multicast protocol for the distribution of search messages throughout the network.

It cannot be assumed that every information required for the discovery and selection of nodes is known beforehand and that appropriate components are included in the middleware. As an extensibility mechanism, the core information service facility must provide an extensibility interface that allows custom modules to add additional information to the local node information record. These modules can be implemented as dynamically deployable Grid services that are automatically instantiated at startup time of the Grid service container (in figure 4.2 these components are represented by the two property info services). Isolation mechanisms introduced later on in section 4.2.5 may be used to shield the local node from malicious activities of such a node information service.

Service discovery in an ad hoc Grid as well as in a regular web service environment can happen for two purposes. Software developers in need of component services for a new application search for certain functionality, or applications already designed to consume a certain service search for available instances of the service. Grid services expose their interface descriptions as WSDL documents like regular Web Services. This technical description - containing details about the operations exposed by a service, the data formats used for communication and protocol bindings as well as contact information such as the communication endpoints - is especially useful for applications trying to identify the right target services and determining the correct invocation methods. Standards for service registries such as the Universal Description, Discovery and Integration (UDDI) standard [131] hold provisions to also include human readable descriptions of the semantics a service offers. Other initiatives try to find ways for describing service semantics in a machine readable way for use in the vision of the semantic web [117]. Neither UDDI nor the semantic web service descriptions have seen a real breakthrough up till now. The primary concern for the service discovery component of an ad hoc Grid middleware is to enable automatic operation of ceratain components of the middleware such as resilience mechanisms. These mechanisms primarily require discovery capabilities based on the technical interface description of a Grid service.

Every Grid service container must maintain a list of services it provides. This list of services must again be exposed to the underlying P2P infrastructure. For MAGE, a service discovery adapter is responsible for either handling search requests from the P2P network or publishing the information into the information space of a structured P2P network. The service discovery component must minimally allow to query an invocation target endpoint address for a Grid service together with the port type of the service. The question expected to be asked most often to the service registrar is: "What endpoint addresses can be used to communicate with a service of a given port type?". DHT structures are most suitable to answer this kind of question. If an excerpt of the service WSDL description is published in the information space using the port type of the service as a key, the aforementioned question can be answered very efficiently. This service discovery P2P adapter is shown in the component overview in figure 4.2.

Access to the P2P discovery functionality is provided through a Grid service interface. The implementation of such a service may open its discovery API for local components to directly access the P2P infrastructure. External clients may use the P2P discovery service on any node connected to the ad hoc Grid network through the P2P infrastructure to perform discovery operations.

4.2.3 Service Communication

Grid services, like web services, have been designed to support remote invocation by sending SOAP messages to the target service. SOAP bindings to other transport protocols can be defined. The most common case is the use of HTTP as transport protocol for SOAP messages. Most web services expose a HTTP URL as a communication endpoint in their service description that can be used to send SOAP messages in the body of a HTTP POST request to the service. In a synchronous invocation, the result is then sent back in the HTTP reply from the service to the client. HTTP has been chosen as a well known protocol that supports the tremendous success of the World Wide Web. With a very clear distinction between client and server in the communication scenario, this is certainly a good decision. First complications to this view are introduced by an asynchronous invocation pattern. In this case, the client just triggers an operation at the service that does not return a result immediately but sends the result in a second message exchange to the client. The connection to the client is initiated by the service. If the client happens to be located behind a firewall or NAT router, sending of the reply message requires the client to poll for the message, since the service cannot initiate the connection to the client.

P2P networks are designed with the basic assumption that every peer in the network must be equally accessible, even behind such barriers. Instead of relying on special implementations of NAT traversal and firewall penetration, the communication capabilities of the P2P network already used for node and service discovery can be leveraged to enable seamless communication between all nodes in the Grid system. Tunneling of SOAP messages through the P2P network allows every node to act as a service provider with equal connectivity provided by the P2P infrastructure. Reliable delivery of messages over the P2P network is required for the transmission of service invocation messages. In practice, this means the definition of a second binding of the SOAP protocol to the underlying P2P transport protocol or the tunneling of HTTP communication through the P2P infrastructure. Such a tunneling solution can either be implemented as an internal communication component of the Grid service container that serializes the invocation messages emitted by a service client to the P2P network and deserializes the same message for invocation on the container side. Or an intermediary component in the infrastructure may handle transmission of the messages. This intermediary component implements an HTTP proxy. The client side component in the communication must be configured to use this proxy for all communication.

Another challenge for such an intermediary component solution is the addressing of messages. Since endpoint addresses within a NAT realm need not be globally unique, a different addressing scheme must be used when communicating through the proxy in order to allow endpoint resolution for the proxy. The address space of the P2P network naturally provides unique addresses and may also be used for identification of the communication endpoints using a proxy. This requires the ability to specify addresses from the P2P network in the client, section 5.2.3 describes a solution that allows to use P2P addresses in an unmodified Grid service client.

Many P2P systems such as the Resource Management Framework (RMF) [77] also provide a publish-subscribe mechanism very similar to the WS-Notification mechanism introduced in the WSRF. This mechanism provides the ability to register clients to receive notifications of changes that happen to a resource managed in the information space of the P2P system. If the WS-Resource documents are managed directly in the P2P information space, registration for and delivery of WS-Notification messages may directly be handled by the publish-subscribe mechanism of the underlying P2P infrastructure. Similar to the tunneling of service invocation messages over the communication mechanism provided by the P2P infrastructure, the registration requests of the client may be intercepted and translated into registration requests of the P2P network. If changes to the resource property document are directly applied to the property document representation in the P2P information space, the notification mechanism of the P2P infrastructure will automatically deliver a notification event to all subscribed clients. Here, another adapter must translate the change event into a WSRF notification message and deliver the event to the event handlers of the Grid middleware.

4.2.4 Hot Deployment

Automatic and non-disruptive service deployment is one of the most important requirements for an ad hoc Grid. With increased adoption of the Grid paradigm, the number of developers wanting to use the Grid will increase, thus giving each developer administrative rights for all Grid nodes is not feasible, especially in an ad hoc Grid environment where personal computers are also members of the Grid. A hot deployment service (HDS) allowing remote deployment of Grid services without requiring the developer to have an administrative account on the system must be available on every node in the ad hoc Grid [81]. A further vital requirement to the HDS is that deployment of a service must be done non-disruptively, i.e. without requiring a restart of the WSRF platform after deployment. This is absolutely essential, since it is not acceptable that every time a new service is installed or an existing service is updated, all other services running in that environment are restarted as well - possibly losing substantial amounts of work in progress. Although this ia just as undesirable as to restart

an entire web server every time a web page is added, currently this is common practice in service-oriented Grid environments. Deployment of a service for the service-oriented ad hoc Grid means installing the factory that is used to create WS-Resource instances that connect a state capturing resource property document to the stateless service implementation.

The following basic operations must be provided by the HDS:

- *Deploy* adds a service to the set of available services on the Grid node. The service is identified by its service name. The operation will not deploy the service if there is a service with the same name already present on the node.
- *Undeploy* removes a service from the node, based on its service name. Running service instances already created are unaffected by the operation.
- *Redeploy* is the chaining of undeploy and deploy. Running service instances are not changed by the redeploy operation, subsequent requests to create new instances will, however, use the newly supplied implementation of the service.

Architecturally, the hot deployment functionality to a Grid node should be offered through a Grid service interface. The standard communication and service access control mechanisms for the Grid apply also to this service, allowing, for example, to apply user and virtual organization management to the HDS. A number of security issues arise from the fact that unknown code may be introduced into a Grid node and executed in parallel to services of other users. These issues and solutions to these issues are discussed in the next section.

The HDS is primarily geared towards use on the nodes of a desktop Grid environment. An implementation of the HDS for a cluster node in the Grid may follow two distinct implementation models. If the nodes of the cluster system are considered single nodes, a Grid service container may be instantiated on every node of the cluster, combining its resources in a multi-node ad hoc Grid environment. Since cluster installations typically use their own local batch queuing systems and the installation of Ad Hod Grid middleware on every node is often not desired by users and administrators, a second implementation pattern for the HDS may be used. In this pattern, a software installation mechanism for the cluster is used to install the necessary applications on the individual cluster nodes while at the same time installing the Grid service that provides access to the cluster application on the head node of the cluster. Then, a cluster is a very powerful raw resource represented by its head node in the overall view of the ad hoc Grid.

While the security and isolation mechanisms presented in the next section may be applied to the cluster nodes in the first design, similar protection mechanisms for isolation of user installed applications are required from the cluster operating system in the second case. Therefore, this thesis focuses on solutions for the first case.

4.2.5 Security

Security is a very important aspect for a Grid middleware. Consequently, a number of proposals and provisions in service-oriented Grid middleware systems exist that regulate access to individual nodes and services and the protection of data transmissions against modification or unauthorized inspection. In the context of this work, the existence of a security mechanism is assumed that governs access to Grid services and WS-Resource instances on every node of the grid. As a rough sketch of such a system, a Grid group may be defined by a group initiator who generates a group certificate. Every new member of the group must receive a personal certificate that is signed with the group certificate. Nodes can determine the group membership of a user by validation of the certificate chain of the group membership certificate. Access to the services provided by a node may then only be granted to users providing a certificate for authentication that has been signed with the group certificate. Access to a newly deployed service may then be restricted to the group of the party that deployed the service or to every group that has access privileges to the node. Access to a WS-Resource instance may either be allowed to all users with access privileges to the node, all members of the group the creator of the instance belongs to, or only the owner of a temporal access certificate that was returned with the resource ID upon creation of the WS-Resource. For the design of the ad hoc Grid middleware, it is assumed that a security module in the middleware can control access to the different services deployed in Grid service container.

The previously described scheme is suitable for all of the application scenarios presented in section 2.2 since the collaborating partners are typically known to each other and form their Grid infrastructure spontaneously on the basis of their previous social relationship. The medical as well as the media science scenario require a middleware solution that handles most of the configuration of the network while access control based on the previous exchange of certificates is acceptable to the users. In the engineering scenario, a somewhat more anonymous yet critical relationship between the computational resource provider and different customers exist. If the Grid platform does guarantee strong isolation between user groups and their services and data, the above solution may successfully be applied in this scenario by creating a distinct group for every resource consumer.

The focus of this work is on the topic of isolation of Grid service instances of different users against each other and protection of the local platform against foreign code. Provisions for isolation are especially relevant to the ad hoc Grid idea, since a number of threats arise from the fact that different users may deploy and run their services concurrently on the same Grid node. For security considerations, a distinction between three different types of applications must be made:

The first type (type 1) is the most modern: a fully service-oriented Grid application programmed in a secure high level language like Java or C#. These

are the easiest to deal with since the virtual machines in which these applications run already have strong security enforcement capabilities built in. However, many scientific applications in Grid computing are based on legacy code bases which cannot be ported to Java for cost and efficiency reasons. These existing legacy solutions are usually wrapped with a number of Java Grid service implementations to make the different legacy components available separately to the service-oriented Grid environment. This creates a major security problem since the native code cannot be constrained by the standard Java or C# security facilities. These service-oriented applications which contain a number of legacy components make up the second type (type 2) of Grid applications. Finally, the third type (type 3) of applications is a monolithic legacy application which is wrapped with a single Grid service but apart from that has no service-oriented attributes and should be considered as a traditional Grid application. With these applications, it is often necessary that users are able to directly access the nodes on which the application runs. These types of Grid applications have all the security problems of the first two types in addition to the security problems incurred by allowing direct access to the Grid nodes.

In addition to this classification of applications, a rough distinction between two types of attacks can be made. A first class focuses on data managed by the hosting environment or by other services, and the second one on abusing other system resources, for instance network bandwith or CPU cycles. Examples for each of the resulting attack scenarios are presented in the following:

- *Data attack against hosting environment:* A malicious service may be used to extract or alter security critical data from the underlying operating system or hosting environment such as the system password files, certificate files or service container configuration. This thread is prominent among type 2 and 3 Grid applications that have a native part that in most Grid service containers is executed with the user rights of the hosting environment, enabling the malicious service to read the configuration of the hosting container, and in many cases even allowing the alteration of configuration files (such as the container authorization lists).

- *Data attack against other services:* A malicious service may be used to read temporary data or results produced by other services as well as input data used by those other services. If, for example, a pharmaceutical company uses a Grid node for computations in the design phase of a new drug, a competitor may deploy a malicious service that extracts the experimental data used as input of the computation or the resulting outputs. Under some conditions, unauthorized access to data of other services is even possible in high level languages like Java that have strong security mechanisms.

- *Resource attack against hosting environment:* A malicious service may implement a *spam bot* that is used to send unsolicited bulk emails from the

Grid node. A number of denial of service attacks also fall into this category. By using native code, an attacker can cause the underlying operating system or hosting environment to crash, evading type safety mechanisms and sandbox constraints of a pure Java environment, effectively performing an internal denial of service attack on the service host.

- *Resource attack against other services:* A malicious service may invoke methods from other services directly or use software licenses for 3rd party software that belongs to other service instances.

For a more detailed analysis of possible threats and solutions in on-demand Cluster and Grid computing the reader is referred to [153]. This work concentrates on a fine grained solution that addresses the threats sketched above, mainly in the context of type 1 and 2 applications. Further analytic examinations and solutions to security problems in ad hoc Grid and cluster computing are presented in [154].

Figure 4.3 shows an abstract view on the sandboxes required to isolate services between different users within the Grid service container (required for type 1-3 applications) and for the isolation of native components required by type 2 and 3 applications. The Grid service container is assumed to be implemented in a safe language providing sandboxing capabilities for components implemented in that language of the Grid service container. Applications implemented using such languages (e.g. Java or C#) are usually executed in a virtual machine. Typically, those languages provide means to integrate native libraries into an application. They do, however, often lack the means to effectively constrain the rights of these native parts of an application to a custom access policy to the underlying virtual machine or operating system as well as to other threads and processes in the virtual machine. An example for this is the Java Native Interface (JNI), that allows access to libraries implemented in C or C++ while the Java sandbox mechanism can only either allow access to such native libraries or block access at all.

For the isolation of different service instances against each other, the following service grouping mechanism is introduced to the deployment service. The basic HDS interface is extended to support *grouped* operations that handle deployment of a Grid service into a secure group in the Grid service container. The groupDeploy operation either takes a service group certificate as a parameter or generate a fresh certificate assigned to the group upon deployment of a service. Subsequent deployment or undeployment operations on the group require that the new service archive is signed with the group certificate or that a signed undeployment request is submitted. This guarantees that only the creator of a service group may add new services to the group or remove services from the group. In addition to the creation of new groups, the *defineGroup* operation may be used to define a service group from already deployed services in the platform.

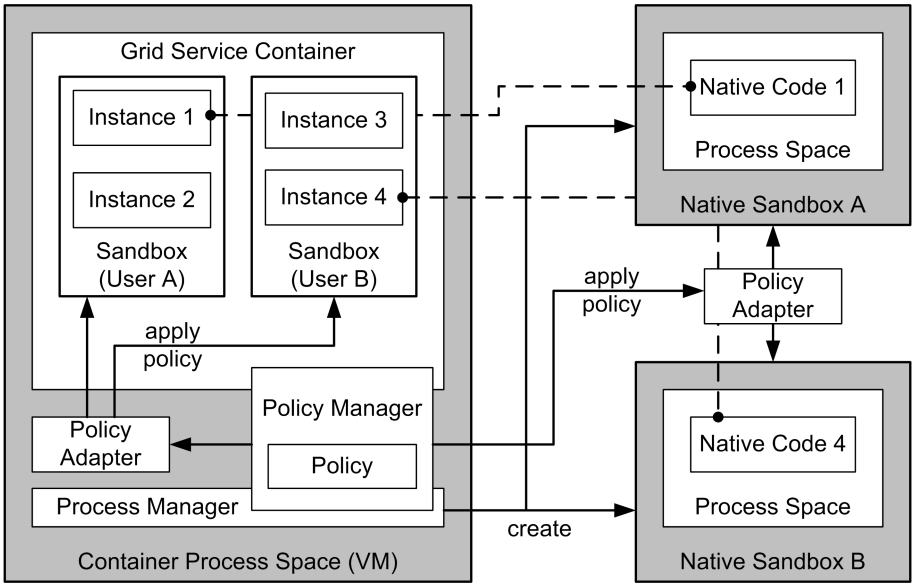


Figure 4.3: Abstract view on the different isolation realms required to address security threads to other users and the Grid node.

All resource instances created by the services in a group are placed in a sandbox. The platform must guarantee that those sandboxes are strongly isolated against each other. If there is a need for communication between different services and resource instances of different secure service groups, regular Grid service communication must be used between the grouped services. This again ensures that access control mechanisms of the Grid environment apply to those inter service communications. Pure Grid service implementations are typically lightweight components that belong to a more complex application, if two distinct user groups require access to the same service implementation, the same Grid service may be deployed twice into the different secure service groups. In order to enforce secure separation of the process spaces of native components of different service instances, the native part of an application must be executed separate from the process space of the virtual machine hosting the Grid service container. This enables the operating system to enforce isolation of the process spaces.

The previously described mechanism realizes a solution to the problem of isolation of the services and resource instances of different users against each other (data attack against other services and to some extent also resource attacks against other services). Security is accomplished by total separation of the sandboxes from each other. The other attack scenarios above require protection of the underlying virtual machine or operating system from malicious activities of services in the different service sandboxes.

Another problem in the confinement of a service arises from the reliance on

native code found in type 2 and 3 applications. Virtual machines for the execution of high level languages like Java and C# typically include a sandbox mechanism, that allows users to specify a policy expressing the access restrictions that should be enforced on the code running within a confined process space of the VM. Similar sandbox mechanisms have been implemented for different operating systems using for example system call interposition [140] and operating system virtualization [23, 51, 166]. Approaches based on system call interposition provide light weight means for fine grained privilege elevation or access restriction. They may more easily be employed to control system access in a flexible policy based way than more traditional Grid approaches that map grid certificates to local user accounts.

Figure 4.3 shows an abstract view on the architectural components designed to enforce such policies to at least type 1 and 2 applications. The owner of the Grid node may specify an access policy to the local system (policies may also be specified per user or per group) that is managed by a policy manager in the Grid service container. The abstract system policy needs to be translated to the different confinement mechanisms available for the virtual machine that hosts the Grid service container and for the sandbox mechanism used to confine native parts of a service implementation.

This confinement infrastructure requires the interception of component access to native components. Such access from a thread within the VM must be intercepted by the Grid service container that instructs a process manager to create a process instance that is confined to an operating system controlled sandbox. Afterwards, every access to the native parts of the service implementation must be delegated to this confined process space. Execution of the native code outside of the process space of the virtual machine and inside an operating system controlled sandbox can protect the Grid node against data attacks to the hosting environment. If the sandbox mechanism of the virtual machine or the operating system also offer the capability to control and restrict resource utilization, this scheme can also be used to protect a platform against both types of resource attacks. An implementation for this protection scheme using Java and Systrace is presented in more detail in section 5.2.5.

Confinement of operating system level processes to a secure sandbox is the most demanding requirement of an ad hoc Grid middleware to the operating system of the Grid node. Absence of such mechanisms in an operating system can prevent the adoption of the ad hoc Grid idea in some application use cases. In the medical and media science use case an approach is feasible that restricts access to the system based on a closed group of previously known users or the restriction to a closed and private network. A scenario more open to potentially unknown users like for the engineering application requires strong protection mechanisms against foreign services running on the same Grid nodes.

Even more open scenarios for the application of an ad hoc Grid environment exist, consider Seti@Home [108] as an example. In this scenario millions of users

installed the application of a single party on their local computers sharing their resources in the search for signs of extra terrestrial intelligence in a large set of data collected from a radio telescope.

A similar deployment of the ad hoc Grid would lead to a situation where millions of users share their computers to potentially allow every other unknown user to install and run his or her application. In such an ad hoc Grid environment, trust is a major requirement for enabling collaboration among the interaction partners, which in our case could be identified as nodes and/or services. Azzedin et al. [21] have classified trust into two categories: identity trust and behavior trust. Identity trust is concerned with verifying the authenticity of an interaction partner, whereas behavior trust deals with trustworthiness of an interaction partner.

The overall behavior of an interaction partner consists of several elements, such as accuracy or reliability. These elements of behavior trust should be continuously tested and verified. In this way, it is possible to collect a history of past collaborations that can be used for future decisions on further collaborations between interaction partners. This kind of experience can also be shared as recommendations to other participants.

Furthermore, the overall decision whether to trust an interaction partner or not may be affected by other non-functional aspects that cannot be generally determined for every possible situation, but should rather be under the control of the user when requesting such a decision. In addition, while the basic functionalities of two applications could be similar, differences in application behavior could be caused by different domain specific trust requirements. As an example, consider the medical as well as the media science sample application. In both scenarios, a number of filter and analysis algorithms need to be applied to a large data set, yet the requirements on accuracy of the returned results and the trust that is required in a node to allow data to be processed by the node are much higher in the medical application scenario.

Therefore, a trust system for a service-oriented grid environment should offer flexible and easy to use components that can be configured to the specific needs of a user on a per case basis. For information about a trust model as well as an initial design for a trust system for the service-oriented ad hoc Grid, the user is referred to [137].

4.2.6 Data Handling

The requirement for flexible data handling mechanisms in an ad hoc Grid middleware was motivated in section 2.3.1. The most important functionality of a data handling component of a service-oriented ad hoc Grid middleware is the ability to transport large amounts of binary data in a Grid service invocation. For the service-oriented ad hoc Grid middleware presented in this thesis, a new approach to handle large attachments in SOAP interactions - Flexible SOAP

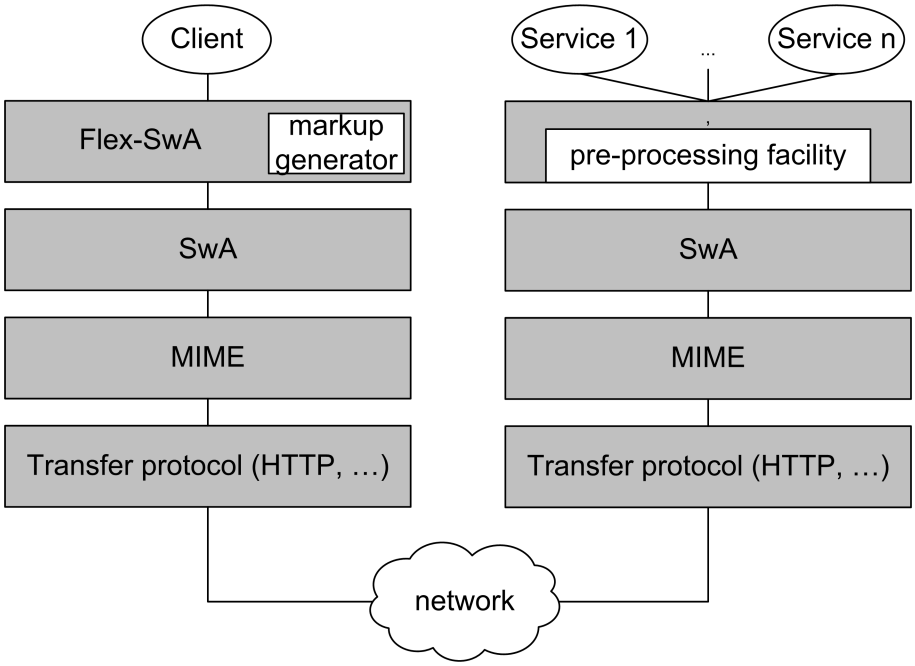


Figure 4.4: Overview of the protocol stack using Flex-SwA.

Messages with Attachments or Flex-SwA for short [97] - is introduced. Instead of leaving the handling of large data attachments to the SOAP engine or the application developer, the Flex-SwA approach offers components for handling of such transmissions in the middleware.

Figure 4.4 shows the entire protocol stack from a service user's as well as the service provider's point of view. The client resides on top of the service user's protocol stack, which is using the Flex-SwA layer to invoke remote services. The Flex-SwA layer uses a markup generator to create an XML description, which refers to the actual location of the large binary data sets and the protocols used to transmit the binary data objects. Such references can directly reference remote files. In case standard SOAP or SwA is used for the transmission of binary data, the application developer must decide whether to provide methods for transmission of data as an attachment or just as a reference and the service application logic is required to handle references, the Flex-SwA architecture covers both cases while providing means to handle direct remote references. Reference elements are either directly embedded in the SOAP body or attached as a one of the parts of a multipart MIME message.

On the service provider's side a preprocessing facility parses markup received from a client and subsequently prepares data transmission. The provider side Flex-SwA layer need not handle data transmission for all markup elements. The unhandled markup can be forwarded to other service providers, thus providing *message forwarding without additional communication cost*.

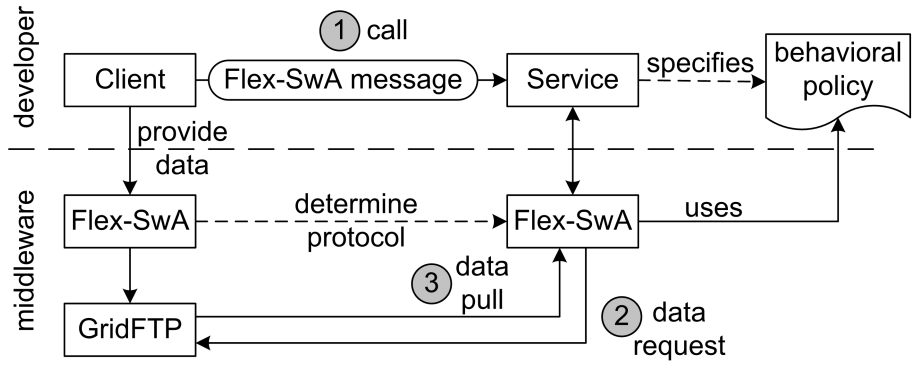


Figure 4.5: Activities performed by the Flex-SwA layer and the client / service during a service invocation.

From an application developer's point of view, Grid service invocation and data transmission remain coupled in a single service invocation operation. The concrete behavior of the platform regarding the handling of data transmission and service execution can be controlled by specifying a *behavioral policy*. As an additional benefit, service developers can use the protocol handling capabilities of Flex-SwA to leverage high performance binary protocols by simply specifying a policy to use them, without having to deal with the protocol details in the application code. Binary protocols can be selected for each message part individually. In contrast to a realization in a more traditional application environment where the developer has to handle every aspect of the communication, most of the functionality needed to handle a specific transport mechanism are realized in the Flex-SwA layer. Figure 4.5 shows the interaction pattern between the client and service during a call using GridFTP [6] for the actual transmission of data. Markup specifying the location of the data is contained in the Flex-SwA message sent to the service (1), the middleware processes the markup and collects the data (2,3) from the GridFTP server for the service. There is no need for the application developer to explicitly handle the GridFTP transmissions in the application logic.

A behavioral policy can be specified as a default behavior for the entire platform (e.g. regarding the selection of a preferred transport protocol) or as a service policy. Two possible behaviors exist regarding the handling of data transmission and execution of a service. In *non-overlapping* mode, the platform performs all data transfer prior to invocation of the service; in *overlapping* mode, data transmission and service execution are performed in parallel. If a service needs to make sure that all data is available on the service provider's platform before it starts processing, it requests the platform to handle invocations in non-overlapping mode. If initialization of the service requires time and is independent of the attachment data, a service developer can specify the service to use overlapping invocation mode, causing the platform to start data transmission and service

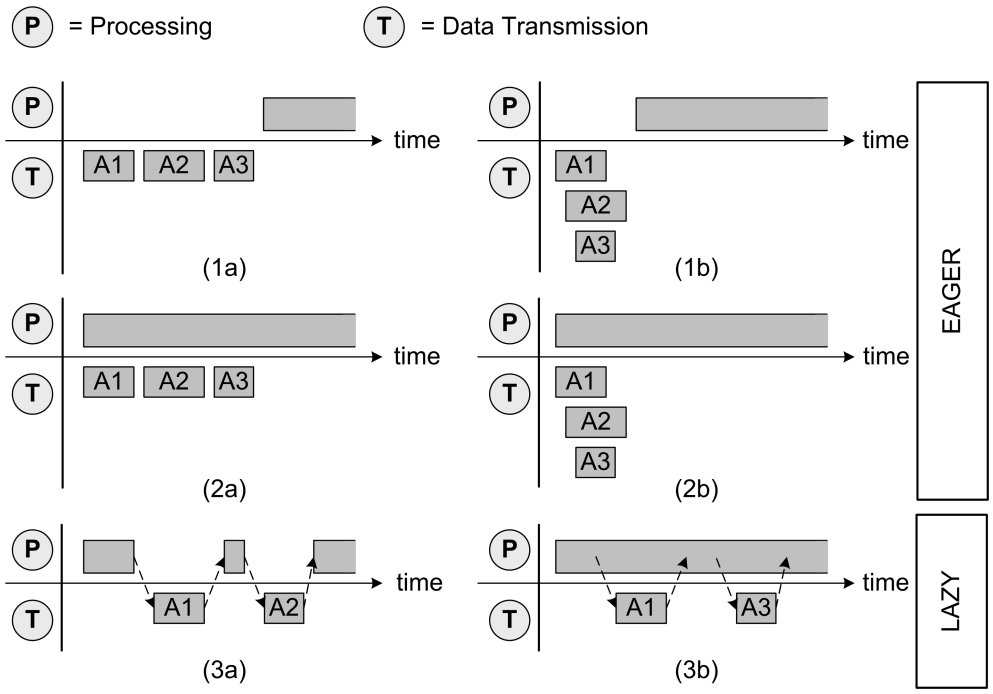


Figure 4.6: Data transmission and service execution patterns.

execution in parallel and thus reducing latency to service execution.

The service can instruct the platform to perform *eager* or *lazy* transmission of attachments, meaning that attachments are collected and stored as soon as possible or only upon a real attempt to access the attachment content. Additionally, the service can prioritize attachments that are tagged to be transmitted in eager mode, leading to a transmission plan for the attachments. The resulting execution and transmission patterns are shown in figure 4.6.

A combination of non-overlapping transmission handling and the eager transmission mode (1a) results in transferring every attachment before the service starts. This scheme is similar to the original transmission via SwA. Transmission of the attachments can also be done concurrently (1b), for example by using several threads, thus providing the possibility of improving the transfer rate.

Combining overlapping and eager transmission handling (2a) results in the immediate start of data transmission and the service. This mode is useful if the service has a certain warm-up time or does not need any attachments at service start. Here again, a concurrent transmission of attachments (2b) possibly provides a better transfer rate than the iterative approach.

Lazy data transmission in combination with overlapping transmission handling results in an on-demand transmission of attachments. If the service needs an attachment, transmission is triggered at that time. Again, there are two modes: *Blocking* (3a) – The service is blocked until data is retrieved from the remote source and stored locally by the infrastructure. *Non-blocking* (3b) – The service

only triggers the transmission and continues directly, transferred data may be accessed by the service upon reception. Blocking mode is used if the service needs the complete attachment before the service can resume execution. Non-blocking mode can be used if only a part of the attachment is needed by the service, e.g. an IDEA algorithm [109] needs the next 8-byte blocks for encryption. The service execution patterns shown in figure 4.6 enable the *demand-driven evaluation and transmission* of binary data.

A critical requirement for the application of Flex-SwA in an open ad hoc Grid environment is interoperability with both clients and service containers that do not support Flex-SwA. There are four possible combinations of SwA and Flex-SwA, as shown in figure 4.7. A service container supporting Flex-SwA has to handle incoming requests from a SwA client just like a standard service container, such that (1) and (2) are non-critical combinations. A client needs a way to find out whether the service supports Flex-SwA (4) or not (3). In the latter case, it has to transmit data as a standard attachment without resorting to other transport protocols. In order to inform the client about the service capabilities, the service description is used to signal the client if a service supports Flex-SwA. While it can be interpreted by Flex-SwA aware clients, standard SwA clients will ignore the extensibility elements that may be embedded transparently enclosed in a documentation element in the WSDL description of the service. If it is an option to break compatibility to clients not supporting Flex-SwA, an additional element may be specified in a WSDL extension. Every Flex-SwA service container supports standard MIME multipart/related messages as protocol for attachment transmission. Therefore, every SwA client can communicate with a Flex-SwA server.

In addition to the information that a service supports Flex-SwA, the client also needs to know which concrete protocols the service supports for the transmission of an attachment. In general, this property is platform dependent and varies over time, i.e. some protocols might only be accessible at a certain time. If the service interface description is generated on request prior to a service call, these protocol capabilities may also be embedded in the WSDL document. The interface definition may, however, be prefetched and cached in the client, depending on the application needs. Therefore, the Flex-SwA platform must offer an additional service that exposes the list of supported protocols directly to a client. For the ad hoc Grid, this information may also be published to the node property set in order to make it directly available through the P2P discovery service.

Some transport protocols used for access to attachment data may require *preparation overhead*. For example, consider the transmission of binary data objects via a GridFTP server that is shared by both service provider and service user. In this scenario, the client has to upload the data to the GridFTP server before the service provider can access the data and retrieve it from the GridFTP server. Other protocols that enable access to the attachment data from the client node do not require such a preparation overhead. Service developers can specify

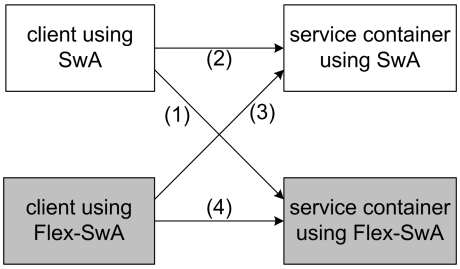


Figure 4.7: Client and server interoperability.

a priority order for acceptable protocols for attachment transmission.

The client calculates the intersection of the sets of protocols that both client and service support. It may then decide to make the attachment data available by any number of protocols in the order of protocol preference expressed by the service. At least one protocol must be supported; the client is free to include other protocols if they, for example, require no or only limited preparation overhead. In order to determine the protocols usable between the concrete pair of client and service provider, the client needs to collect the supported protocols of the platform prior to the service invocation. For this purpose, a list of supported protocols may be included in the WSDL interface definition that the service provider platform emits.

Figure 4.8 shows the basic architectural components required to enable Flex-SwA interaction between client and service provider. The client creates an *outport* for the attachment of large binary data and requests a reference object from the Flex-SwA layer that may be included in the service invocation message. The protocol to be used may either be specified by the client or the platform resolves the protocol information for a concrete service invocation by requesting a service description and the prioritized list of supported binary protocols from the service provider. A Flex-SwA handler in the request processing chain for service descriptions queries a protocol manager for this purpose at the service provider side, that returns the list of possible protocols after receiving policy information for the service from a policy manager. Markup for the attachment is generated in the request handling chain of the client and added to the request message. Upon reception of such a message, the preprocessing handler of the service provider consults the policy manager to determine the transmission patterns to be used in the actual handling of binary attachments. An *inport* object is then passed to the service to allow actual access to the attachment data. Transmission of the data is handled by protocol handlers on both sides of the communication. Through the protocol handler mechanism, the ad hoc Grid middleware can use protocols that require static infrastructural support like the availability of a GridFTP server together with more flexible protocols such as data transmission or even data distribution protocols based on a P2P network. Through the definition of appropriate behavioral policies, the platform can flexibly be configured to work well

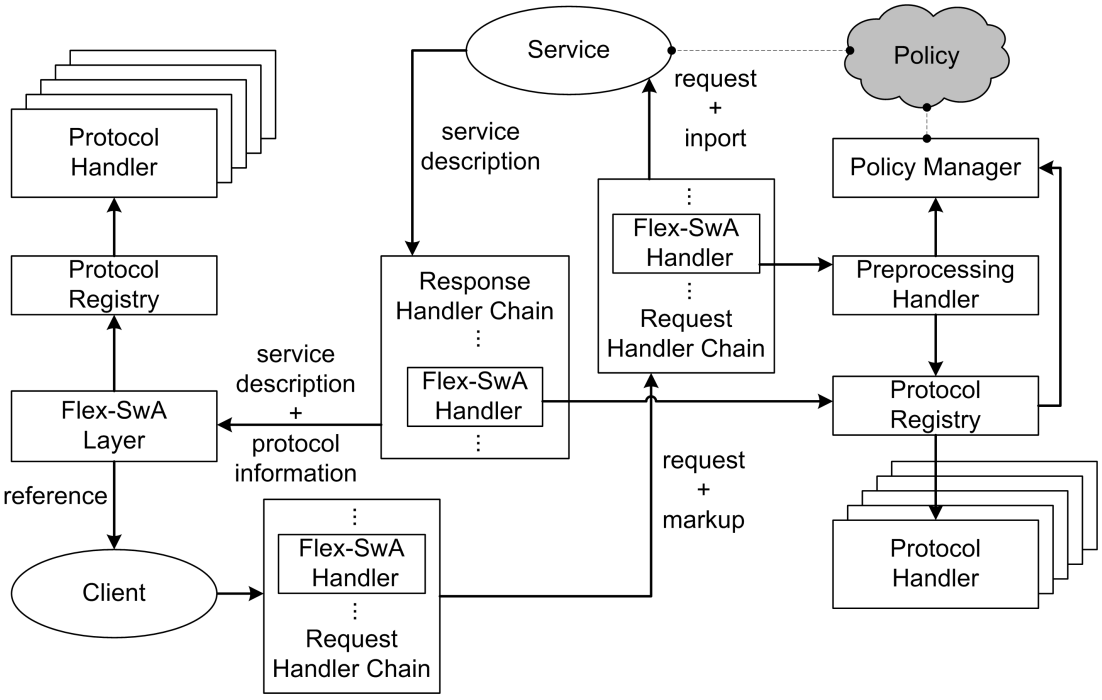


Figure 4.8: Overview of the architectural components of the Flex-SwA infrastructure.

in both environments without having to change application specific transmission logic or binding of an application to one particular protocol.

The abstraction of data transmission to the use of outports on the client side of the communication and inports on the service provider side also allows to implement synchronous transmission protocols between client and service. In this scenario, the Grid service invocation message is used to establish a connection between an outport and an inport through an external channel. The service can then read data from the inport while the client produces data and transmits it through the outport. This interaction pattern may be used to create WS-Resource instances that are chained through inport-outport channels that use a streaming protocol for transmitting data. Those WS-Resource instances may continuously process data as an online filter chain, e.g. in a multimedia application. For more details on this synchronous parameter transmission mechanism the reader is referred to [118].

4.2.7 Grid Process Execution

The Business Process Execution Language for Web Services (BPEL) has been accepted as the de-facto standard as the definition language for Web Service based Business Processes. A BPEL description forms the blueprint of an actually executed process that is usually controlled by a process execution engine. A

process instance captures the state of execution of a potentially long running process. BPEL offers the ability to describe data containers (variables) using XML Schema as their structural definition language and the internal data flow of a process. Basic activities in a process definition represent, for example, the reception of input data from a user or another process and the synchronous or asynchronous invocation of external services. Finally, BPEL defines primitives such as *sequence*, *while*, *switch* and *pick* for the specification of control flow within a process.

Emmerich et al. conducted experiments that showed the applicability of BPEL as language for the definition of scientific workflows [58]. For its popularity and the resulting support from both open source developers and companies developing execution systems and tools for the BPEL language, it is chosen as the basic language for the execution of Grid processes. This decision follows arguments towards use of a standardized process execution language for application on the service-oriented Grid also presented by Yu&Buyya [184] and Leymann [113].

Execution Environment

A standard approach to the implementation of a BPEL process execution system is the implementation of a process execution engine as a server component. Grid nodes in the ad hoc Grid usually run a Grid service container such as Tomcat that is also capable of hosting a BPEL execution engine implementation. This BPEL execution engine provides access to the deployed BPEL processes by exposing a web service interface to the process. The engine architecture does not by itself contradict the ad hoc Grid idea, it is possible to deploy such an engine on every individual node for use of the node's owner or to provide process execution support as a service to other users. Since the process execution engine holds information about the execution state of typically long running processes, it should be provided by reliable nodes in an ad hoc Grid.

Other approaches for the P2P based execution of business processes such as [33] try to counter scalability and robustness concerns in centralized implementations of a process execution engine. However, they have not found widespread adoption. With the ability to deploy a process execution engine on every node in the ad hoc Grid, scalability regarding the number of users of an abstract process description can be achieved. Even though failure of the process execution engine is more catastrophic, the probability for this is much smaller than the probability of failure of individual component services of a process. This latter case is addressed by a robust process execution component in the MAGE middleware. Therefore, an engine based approach was chosen for the initial design of the MAGE middleware, leaving the ability to easily replace the engine based by a pure P2P approach in the future.

Figure 4.9 shows a design overview of the components involved in the integration of a business process execution engine. The engine is supposed to contain an

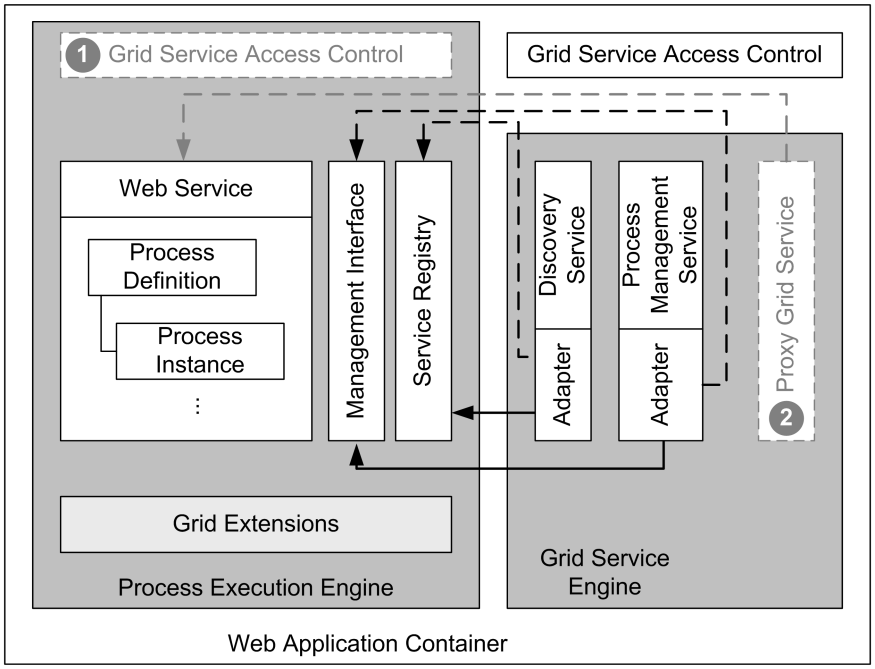


Figure 4.9: Integration of the ad hoc Grid middleware and the process exection engine in a single web application container.

implementation of the Grid specific extensions to the BPEL language described later in this section. Purpose of a process execution engine is the creation and management of process instances from a process description and the provision of access to these process instance through a web service interface. The web services that represent a business process are usually registered in an internal web service component that also exposes them to external partners. Additionally, a process execution engine is assumed to provide some form of a management interface for processes that at least provides the ability to deploy and remove process descriptions and additionally may offer information about the internal execution state of the engine. The internal management interface of the process engine is used by a corresponding process management Grid service, that offers support for deployment and removal of process descriptions in the Grid middleware. Information about the deployed processes and the corresponding web service instances and endpoints is provided through the discovery service of the Grid middleware. The ability to easily replace the engine implementation while reusing the external interface implementation of the management service can be guaranteed by implementing access to the internal management and registry components of the process execution engine through adapter components of the Grid management and information services. If no internal interface to the management and service registry functionality of the process engine exists, adapters may be used that connect to the external interfaces of these services.

Implementing the management and information services for the process execution component as Grid services has two purposes. Firstly, it guarantees that Grid processes are discoverable using the standard discovery mechanism of the ad hoc Grid middleware. Secondly, the standard Grid service management functionality and access control mechanisms of the middleware apply to this two Grid services. The management and information interfaces of the process engine can be restricted to only allow local access, protecting them from unauthorized external use.

The architecture introduced above offers two possible alternatives to extend service access control mechanisms from ad hoc Grid middleware to the services provided by the process execution engine. Firstly, an implementation of the Grid service access control mechanisms may be included in the implementation of the process execution engine. Secondly, transparent proxy services for the processes may be generated and exposed by the Grid service execution engine. In this case, access to the process execution engine can be restricted to the Grid service engine using the access control mechanisms of the process execution engine or the web application container while the Grid services corresponding to the deployed processes are still managed by the ad hoc Grid middleware. The two alternatives for the access control mechanism are displayed in light gray and numbered (1) for the extension of the Grid security components and (2) for the generated Proxy Grid Services in figure 4.9.

In section 2.3.1, the requirement for support of features of the underlying Grid infrastructure by the process execution component was introduced. To better address these requirements, extensions must be specified and implemented in an execution system for BPEL processes. In the following, extensions to the standard BPEL language are discussed that better address these requirements.

Language Extensions

The current BPEL 1.1 standard lacks fundamental constructs, required to implement and control the parallel invocation of multiple Grid services without unnecessarily complicating the process description. A recurring pattern in the sample application scenarios presented in section 2.2 is the parallel application of an algorithm to an input data set. From an abstract point of view, the parallel operation requires the following three steps:

- splitting of the input data into smaller parts
- application of an algorithm either implemented as a single Grid service or as a more complex composition of basic services to each partial input
- collection of partial results from the parallel invocations and subsequent merging into a single result

The parallel invocation of a Grid service requires knowledge about the services to use in the invocation. Partner services for invocation in a BPEL process must be specified using a `partnerLink` definition in the process. The partner link captures the port type of the target service for the invocation as well as the target endpoint address used to exchange messages with the service. The concrete target endpoint addresses of a service can be discovered and assigned dynamically to a partner link within the process. For the parallel execution of n invoke activities n distinct partner links (and n distinct invoke activities in a flow structure) must be defined for the process in standard BPEL. Depending on the degree of parallelism of the application, this leads to a large number of partner link and invoke definitions that blow up the size of the process description.

The *GridForEach* (GFE) construct is introduced as an extension to the BPEL language to capture this basic parallel execution pattern in a single construct. Mandatory child elements for a GFE element in a process description define external services that split input data (`gridSplitterInvoke`) and merge result data (`gridMergerInvoke`). The actual operations to be carried out in parallel are specified in a nested `scope` element. The nested scope of a GFE can contain any structural BPEL definition such as flow or sequence.

The inner scope of the GFE references a number of partners. Links to those partners may either be static links or dynamically created partner links. A static partner link is constant among all instances of nested scopes. A partner link set must be defined for the GFE element that captures the dynamic partner links to be used in the nested scope. Each partner link to be used in the parallel execution of the nested scope must only be defined once for the process as opposed to standard BPEL where n partner links must be defined for a parallel execution.

The input variable of the GFE is also the input variable to the splitter service, its type defines the parameter type of the operation specified to be invoked for the splitter service. The output of the splitter service is a collection of individual data structures. The maximum number of instances of the internal scope created by a GFE implementation in parallel is the minimum of the initialized partner link sets and the number of elements in the splitter output collection. The GFE assigns the data contained in every element of the splitter output collection together with a partner link set to a different instance of the nested scope. If the number of splitter output elements exceeds the number partner link sets, creation of new scopes is deferred until partner link sets become available. Partner link sets may become available for two reasons: Either execution of a scope terminates and releases the partner link set, or all required entries for a partner link set have been discovered or created.

Individual results calculated in the inner scopes may either be handled by the process execution engine or by the merger service. In the first case, the result variable of the inner scope of every operation is collected in an engine managed result collection, this collection is then passed to the merger service that returns the combined result. In the second case, the merger service must implement

two operations: the first operation handles the addition of a new result to the overall set of results, the second operation is used to collect the merged results. While the first merger may be implemented as a stateless web service, the second service must maintain state information during invocations and should therefore be modeled as a stateful WS-Resource. In both cases, the data type of the return value of the merger operation matches the output variable type defined for the GFE in the process.

Partner link sets collect the dynamic partner links associated with a nested scope. The definition of a GFE element can include the reference to an initialization service that may perform any form of complex initialization or maintenance of the partner link set associated with the GFE. If the initialization service is omitted from the specification, the following standard mechanism should be used by the engine. Either form of initialization service may be specified to be executed *once* before processing of the split data is performed or *continously* during split data processing. Continous execution of the initialization service can greatly improve robustness of the parallel execution. If a Grid node fails during the execution of a nested scope, the instance of the partner link set is removed from the collection used for input processing. In this case, a new resource factory may be discovered by the initialization service and used for the parallel execution of the nested scope. BPEL offers the ability to define fault handlers for a scope, the same is true for the nested scope of a GFE. If the fault handler associated to the nested scope cannot handle an error, the fault condition is automatically escalated to the enclosing scope (the GFE element) that reassigns the splitter result collection element to the unhandled elements and uses another partner link set to handle the element.

A partner link set can contain regular partner links or resource links. Resource links are another Grid specific extension to BPEL, combining the partner links for a WS-Resource instance and for the factory capable to create such an instance. For every resource link in the set, the P2P discovery service is used by the engine to find an appropriate factory service that can create an instance of the WS-Resource. The engine performs discovery of factory services until an upper bound of partner link set instances is reached or until the number of entries in the splitter result set is reached. The upper bound for partner link sets is a property of the set definition. Every resource link has two properties assigned, `preCreate` and `postDestroy` that govern handling of the resource creation and destruction between assignments of a partner link set to a nested scope instance. Possible values for both properties are `once`, `scope` and `never`. Their meaning is that creation or destruction of the resource happens once for the entire GFE construct, once for every scope instance associated with the partner link set or never (i.e. resource handling is explicitly encoded in the process). In the latter case, the nested scope instance is responsible for controlling the resource creation and management. This mechanism also allows to initialize a partner link set in one GFE element in the overall process and reuse the set in another GFE.

A symbolic node name may be defined for every resource link definition. The engine then ensures that the resource factory for all resource links with the same symbolic node name are located on the same Grid node. This feature is useful to ensure that different resources in a sequence are collocated on the same Grid node. As a usage scenario for this feature, consider the processing of multimedia-data by passing the data through subsequent operations where transmission of the data between nodes may be expensive.

Figure 4.10 shows an excerpt from a sample process that implements distributed video cut detection (only the definition of static partner links and variables and some attribute details are omitted from the process). The process engine will use 5 nodes in parallel. Allocating the resource instances happens once before applying the inner scope to the splitter result, resource instances are re-used between different executions of the nested scope. The input to the GFE construct is the reference to an MPEG video, the output is a complex data structure containing the extracted features of the input video. Invocation of the discovery service for initialization of the partner link set is implicit, the platform can determine the required factory port types from the partner link specification of the resource link.

Another Grid specific construct that needs to be reflected in a process description is the ability to register for the reception of notifications from a WS-Resource. A Grid node automatically emits a notification message to any client instance that previously registered itself as an interested party in the change of the value of certain WS-Resource properties. A typical use case in a process specification will be the invocation of an operation on a WS-Resource which has a runtime that exceeds the network timeout for a synchronous invocation. The process may then register for notification on a certain result value of the WS-Resource and suspend execution of the process until the corresponding notification is received from the WS-Resource signaling completion of the operation. Two elements `registerForNotification` and `receiveNotification` are introduced as an extension to model this operational pattern in a Grid process. Even though registration for notifications can be implemented by direct invocation of a Grid service method on a WS-Resource and reception of the notification could be reflected using the `receive` activity of the BPEL language, the two elements directly encapsulate correlation of process instances and the concrete WS-Resources. The purpose of the elements in the process specification does more clearly express the real intention of the operations.

The primarily intended use of the GFE construct is the direct instantiation and invocation of WS-Resources on a Grid node in the ad hoc Grid. The implementation pattern shown in figure 4.11 may also be used to schedule processing jobs for the individual splitter result elements on a cluster node. The pattern assumes the availability of a factory on a cluster head node that creates a resource instance representing the job that is submitted to a cluster queuing system. Service-oriented Grid middleware systems often provide generic services,

```
<partnerLinkSets>
 <partnerLinkSet name="plsA"
  maxInstances="5">
  <resourceLink name="rlA"
   preCreate="once"
   postDestroy="once" ... >
   <factory partnerLink="cutsFactoryLink"/>
   <resource partnerLink="cutsLink"/>
  </resourceLink>
 </partnerLinkSet>
 <partnerLinkRef partnerLink="..."/>
</partnerLinkSets>

<gridForEach
 partnerLinkSet="plsA"
 scopeInputVariable="cutsInput"
 sequentialMerge="yes" ...>

 <gridSplitterInvoke partnerLink="splitterLink"
  operation="split"
  inputVariable="gfeInput"
  outputVariable="splitterSet" .../>

 <gridMergerInvoke partnerLink="mergerLink"
  addResultOperation="addResult"
  addResultInputVariable="cutsOutput"
  collectResultOperation="collectResult"
  collectResultOuputVariable="gfeOutput" .../>

 <scope>
  <sequence>
   <gridInvoke resourceLink="rlA"
    operation="findCuts"
    inputVariable="cutsInput"
    outputVariable="cutsOutput"/>
    ...
   <registerForNotification .../>
   <receiveNotification .../>
  </sequence>
 </scope>
</gridForEach>
```

Figure 4.10: A sample application of the Grid-For-Each construct. The process uses 5 instances of a video cut detection service (cuts) in parallel to process an input video.

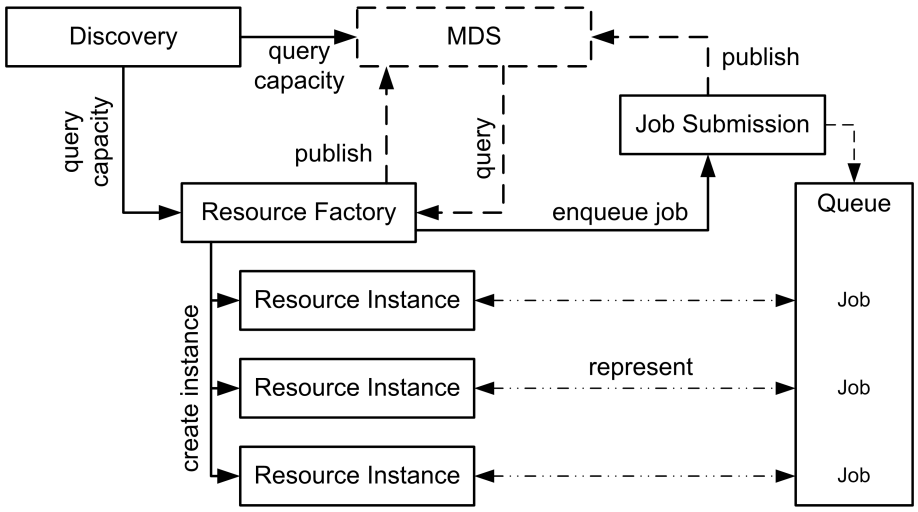


Figure 4.11: Pattern for the creation of multiple jobs in e.g. a cluster queue.

such as the `ManagedJobFactoryService` of GT 4, that implement such a factory generically for job submission to the cluster. The WS-Resource instance representing a job in the queue can be used by a client to gather information about the execution state of the job and collect results of the execution.

Assignment of the individual splitter result elements to a Grid node is a scheduling activity that requires further information about the ability of a factory to create further resource instances. This information may either be collected directly from the factory or be published to the standard monitoring and discovery service.

The following basic scheduling scheme should be implemented by the default initialization service for the GFE. By default, only a single resource instance is created on every node a resource factory was discovered on. In the factory part of a resource link, a property of the factory may be specified that represents the number of additional resources the factory can create. In cases where this simple scheduling scheme is not sufficient, a custom partner link set initialization service may be implemented and specified for the GFE element that performs more complex scheduling tasks. The overall mechanism provides the flexibility required to be integrated with a cluster oriented Grid environment as well as with an ad hoc Grid environment comprised mainly of desktop computers as its nodes.

4.2.8 Integration of the Infrastructural Components

The data handling mechanism introduced in section 4.2.6 is especially useful for the description and execution of Grid processes by an execution engine. Typical Grid services require large amounts of input data and potentially create large

amounts of output data in a more complex application scenario. In a more classical Grid application environment, the task of handling the data is left to the Grid application developer. The application logic must contain logic that explicitly manages staging of input files and results through management interfaces for file transmission protocols. Using the data handling mechanisms presented in the previous section, input and output data of a service may directly be passed by reference through the Grid process execution engine. The actual transmission of the data between individual services is handled transparently by the underlying infrastructure through protocol handlers. The mechanisms for the specification of node affinity of a resource in the GFE construct may be used to help the underlying data handling infrastructure by reducing the amount of data that actually needs to be transmitted over the network. Internal data handling (i.e. copying between variables within the process) is non-critical since no bulk data is passed through the process execution engine when the data movement component is consequently used for partner services, only relatively small references are passed through the process execution engine.

The P2P discovery service and the hot deployment feature of the ad hoc Grid platform may be used by a process to prepare the execution environment before or during the execution of a GFE activity. After discovering a number of nodes that do not provide the required factory services (i.e. can not support the resource link type required in the nested scope of a GFE), the process can actively deploy the factory if it has access to the Grid service archive. The most adaptive and eager implementation of a parallel execution process combines continous partner link set initialization in the GFE activity with concurrent and continous execution of a discovery and deployment sub-process. Such a process automatically deploys a resource factory to every newly arriving node in the ad hoc Grid, which is then automatically included in the set of resources used by the GFE construct to process the input data set.

The ability to support certain transmission protocols as well as the bandwith available for further binary transmission may be of great importance to a scheduling system trying to determine the right Grid node to deploy another service to or to request the creation of a new resource instance. Therefore, the data handling component should use the node information service to publish both informations and make them available to the P2P discovery service.

4.3 Development and Management Environment

Support for the development of Grid applications is an important aspect of an ad hoc Grid environment. Tools should support application development in a way that enables rapid application development without delay caused by dealing with

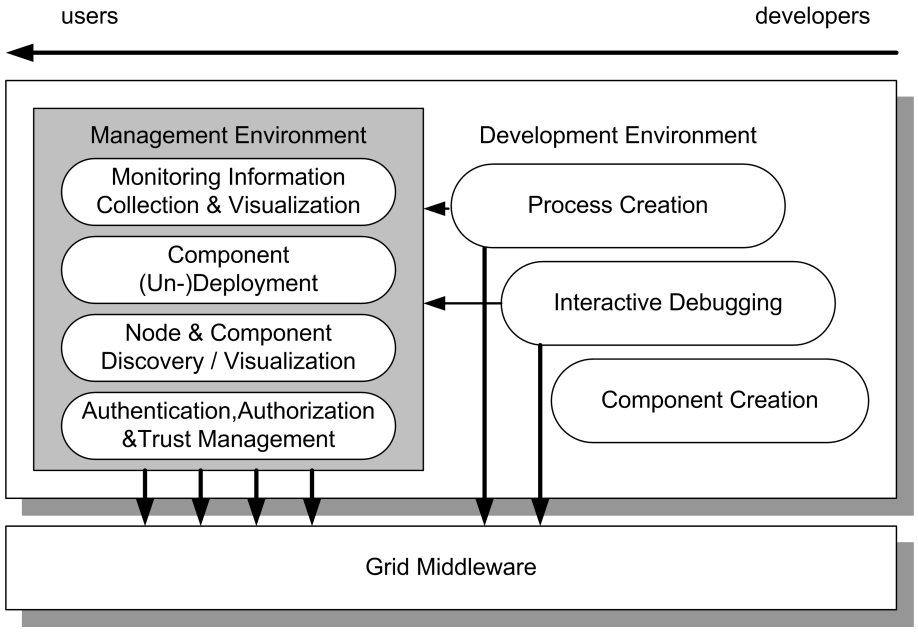


Figure 4.12: Components in the ad hoc Grid development and management environment.

overly complex middleware issues. Integrated development environments offering a wide range of interactive development functionality have been widely accepted as tools improving for example development time and software quality. Many software developers facing the task of Grid application development will already be familiar with using IDEs on a daily basis. Grid development tools should therefore be integrated into common IDEs to support developers with experiences in different application domains to move into Grid application development.

Application development in the service-oriented Grid can happen on different levels of abstraction. The Grid middleware and Grid solution providers may offer basic building blocks for the component-oriented development of larger applications, but creation of a Grid application solving a particular problem may also be the composition of available services into a complex choreography of services and resources. This component-oriented approach offers the potential for domain experts in an application domain to develop large scale distributed applications even though they possess only limited knowledge about the details of Grid middleware. In this sense, the Grid users become solution producers themselves and usage of the Grid really becomes problem solving by applying component-oriented software development. This process can be regarded as the task of identifying the workflows required to align the solution components in order to allow them to process a set of input data. This process is often a collaborative task between different experts. From this notion, we derive the need for collaboration support in the process development tools of a Grid development environment.

This approach is only feasible if the development tools offer sufficient support to hide the middleware complexity from the domain experts, allowing them to concentrate on their primary field of expertise rather than bothering with Grid middleware specific questions. A model-driven development approach is a good approach to support tackling the inherent complexity of the Grid as an application environment.

While interactive development support is a desirable goal, many software development practices, such as automated testing of software and automated build environments, require tools for Grid development to not only support GUI based interactive development. Such tools must also offer the ability to use them in command line oriented environments or automated build systems without a graphical user interface.

Figure 4.12 shows the basic components for an ad hoc Grid application development environment. Arrows indicate direct access to the Grid middleware in terms of middleware control or use of the communication facilities (e.g. for collaborative editing of Grid processes). The component ordering from right to left expresses their association with the user in roles ranging from a developer to a Grid user. Notice also, that the ad hoc Grid management environment is usable on its own but also as a part of the Grid development environment, offering additional functionality for interactive testing and debugging purposes (i.e. deployment of Grid services under development).

4.3.1 Model-Driven Development

Model driven architecture (MDA) [134, 122, 31] has been proposed as an approach to deal with complex software systems by splitting the development process into three separate model layers and automatically transforming models from one layer into the other:

1. The *Platform Independent Model* (PIM) layer holds a high level representation of the entire system without committing to any specific operating system, middleware or programming language. The PIM provides a formal definition of an application's functionality without burdening the user with too much detail.

2. The *Platform Specific Model* (PSM) layer holds a representation of the software specific to a certain target platform such as J2EE, Corba or the service-oriented ad hoc Grid middleware.

3. The *Code Layer* consists of the actual source code and supporting files which can be compiled into a working piece of software. In this layer, every part of the system is completely specified.

MDA theory states that a PIM is specified and automatically transformed into a PSM and then into actual code, thus making system design much easier.

The trick, of course, lies in the development of generic transformers capable of generating the PSM and code layers from the PIM [67, 172]. The reverse transformation from the code layer up to the more generic platform specific or independent layers is called *architecture driven modernization* (ADM) by the OMG. Since service-oriented Grid computing is a young field, there are almost no model driven development tools and consequently existing Grid applications have usually not been created using model driven development. In fact, many existing Grid applications rely on not even service-oriented Grid middleware. Therefore, the ADM direction of transformations from existing code into increasingly abstract model representations is a very useful feature for solution providers and application developers.

Having an abstract model provides the ability to gain a view on a complex software system that allows to reduce complexity in certain areas, leading to the isolation of certain practices or main areas of interest. The higher level of abstraction also allows the creation of a library of patterns and best practices, allowing solution providers to offer tools that better support application developers in their tasks by providing well known recipes for common problems.

The model driven development approach should consequently be used for the Grid development tools that are part of the ad hoc Grid middleware. For more detail on a refinement of the MDA layered approach the reader is referred to [157, 158]. This approach was proposed in order to allow for a further separation of concerns between developers focusing on the application domain and developers focusing on Grid specific application development. The resulting model stack is depicted in figure 4.13. In the MDA definition of the model stack, there is no additional separation of the platform specific layer. The term *business layer* refers to the "business" or application logic of a service.

To facilitate the separation of the PSM layer, a UML Grid Profile is introduced to model the Grid concerns in the business layer. Figure 4.14 shows the Grid Profile metamodel. The profile consists of three stereotypes: GridService, GridMethod and GridAttribute. GridService can be used to mark a class, GridMethod an operation and GridAttribute an attribute. This is similar to the process of marking PIM classes to prepare them for transformation to a PSM suggested by the OMG [122]. But unlike the OMG approach, the developer does not place the marks in an invisible PIM add-on layer, but into the upper PSM layer. This leads to greater clarity, because no invisible components influence the transformations, and the PIM layer stays truly system independent.

The separation of concerns between the upper layer platform specific model representing the *Grid* as the target platform, and the lower layer PSM capturing details of an implementation for a concrete target system (i.e. a Grid middleware platform such as GT4 or Unicore/GS [173]) further supports the creation of Grid development tools with a high level of reuse. The Grid profile meta-model represents the general concepts of service-oriented Grid computing introduced by the WSRF [132]. The GridService corresponds to the service implementation of a

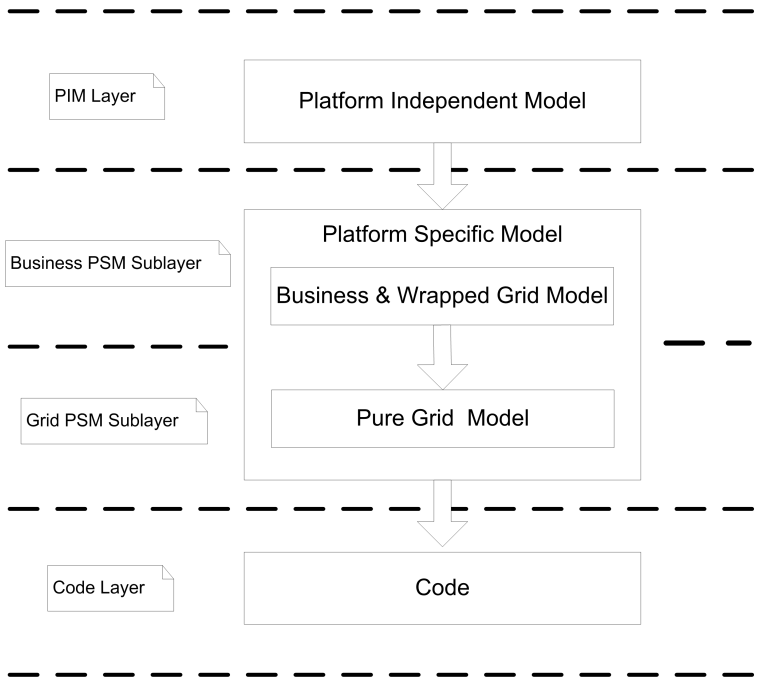


Figure 4.13: Revised model stack with separation into upper and lower PSM Layer.

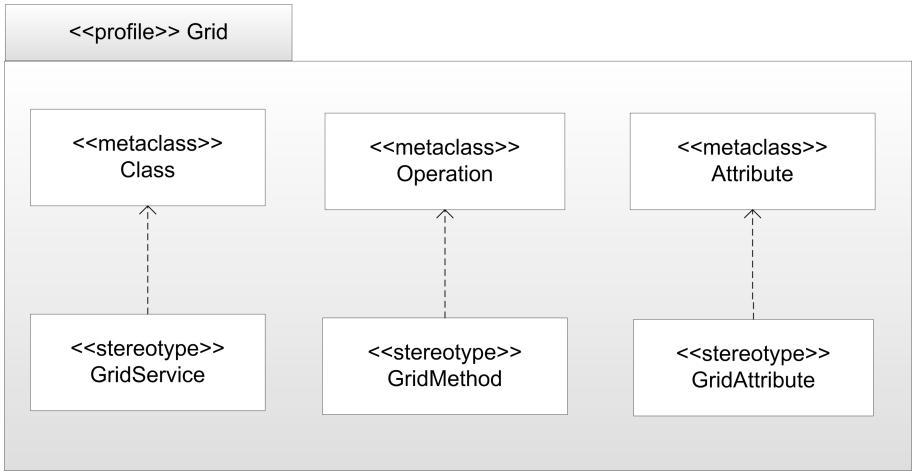


Figure 4.14: Metamodel of the UML Grid Profile.

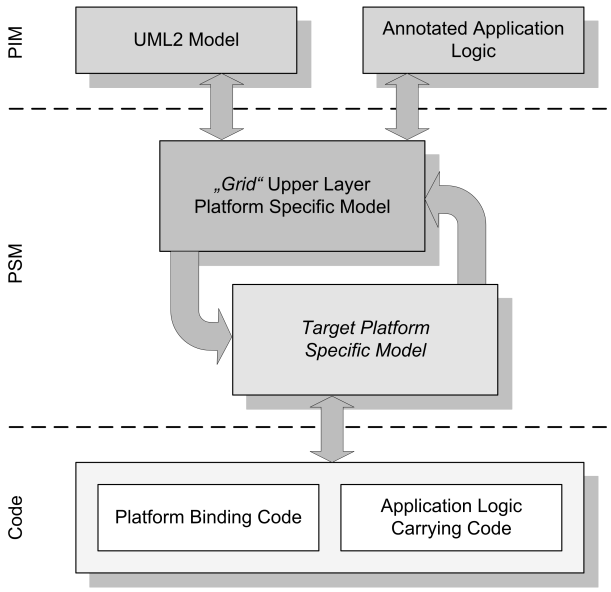


Figure 4.15: Relationship between different (meta-)models and source code in the service creation component of the Grid development tools.

WS-Resource with its methods (GridMethod) available to act on the properties collected in the resource properties document (GridAttribute). The representation of a service oriented Grid application using these upper layer modeling elements is independent of any specific implementation of the WSRF standard. Similarly, the concrete implementation of such an application can be separated into two sub-layers representing the core application logic on the upper-layer and target platform binding code corresponding to the lower layer platform specific model of the Grid applications. A model driven development tool for the service-oriented Grid should separate it's internal representation of a Grid application on the platform specific level.

4.3.2 Service Creation Support

A first step towards support for rapid development of Grid applications is a component allowing fast creation of Grid services as the basic components of a service oriented Grid application. Figure 4.15 shows the relationship between different model representations and the actual platform specific source code for such an application. In a typical model-driven software development approach, software architects start by modeling an overall application in a platform independent manner, often by creating an UML model of the software (upper left corner). This model can relatively easily be transformed into a Grid platform specific model. By using the UML Grid profile presented in the previous section, developers can directly model the upper layer PSM of the software using stereotypes

for the different components and aspects of their solution.

The step from a Grid PSM to a target platform specific model can in many cases be automated or at least be supported by development tools. Strong separation between target platform specific binding code and application logic carrying code makes the mapping from the lower layer PSM to the actual source code a straightforward mapping that can be fully automated. Many integrated development environments offer facilities for the specification and execution of such trivial mappings.

Actual source code carrying the application logic is often not fully generated from a pure model but rather attached to the structural model of the application and carried on through the different model transformations to the lower level model representations. As an alternative path from the platform independent model, an application may be annotated and used as the input to a transformation tool that generates a platform specific model for the Grid that can then be further transformed into the actual implementation of the Grid service for a concrete application, carrying the original application logic into the Grid enabled implementation. Development tools using this abstract architectural model should also be capable of transforming a specific implementation upwards into more abstract representations while keeping the binding between model elements and their implementation.

Figure 4.16 shows the different components used to represent and transform models throughout the Grid development tools. The gray "MT" circles denote general model transformation tools. Since they are applied to model representations to be developed or based on a common meta-model language, they are likely to be implemented using a model-to-model transformation framework for that meta-model language such as ATL [104]. The transformation from model to source code and vice versa is modeled using distinct *Code Emitters* (CE) from model to source code and *Code Interpreters* (CI) from source code to model. Since the annotated application logic carrying code is a source for upper layer model information, a direct relationship (path in the model transformations) exists between the upper layer PSM and the source code (shown by the dashed lines).

The transformation from the upper layer PSM to the target platform specific lower layer PSM can often be fully automated using common patterns for the lower layer PSM model (an implementation for the MAGE platform as the target platform is presented in section 5.3.1). To support another target platform implementation, only this target platform specific mapping model must be replaced. All the common infrastructural components enabling the transformation, all components bridging the gap to higher level model representations and tools can be reused, using the reverse mapping from actual implementations to model representations. The availability of such target specific mapping models allows easy porting of Grid component implementations to another target platform. In this case, a developer starts from an application for one target platform,

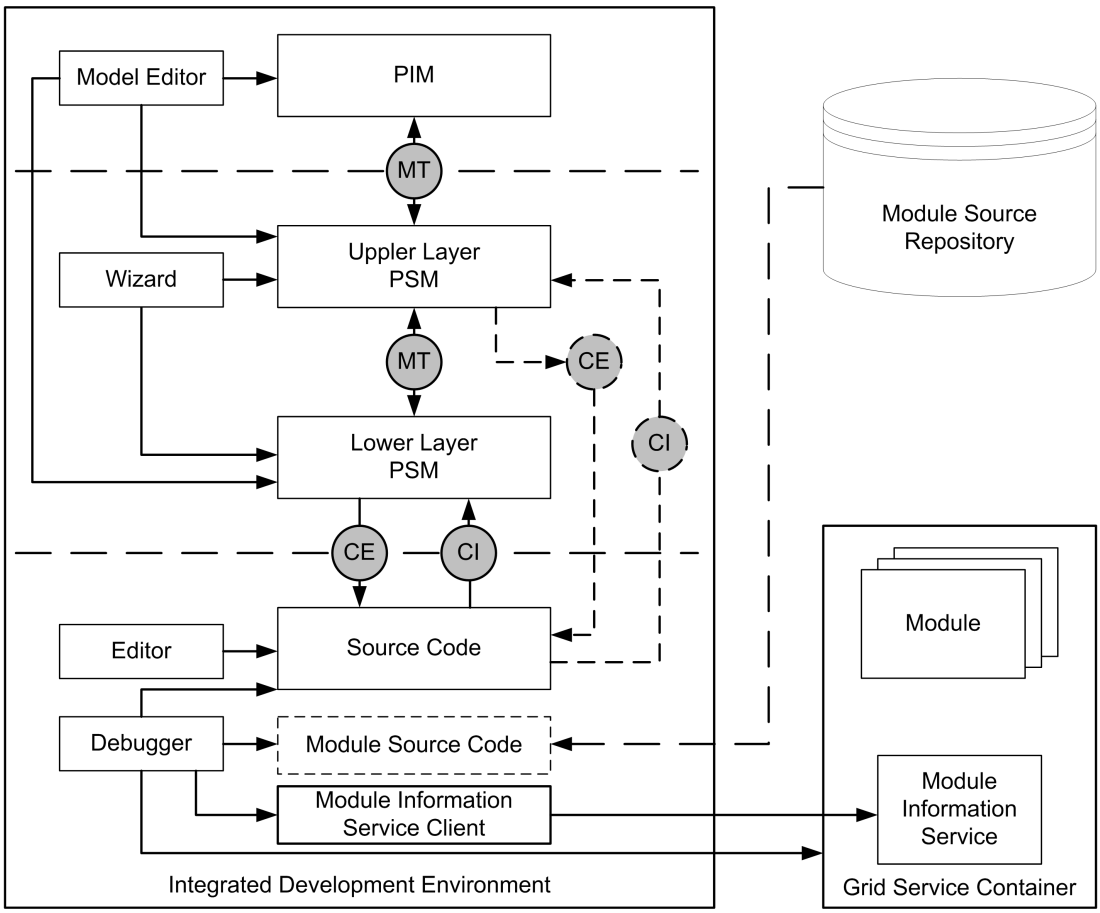


Figure 4.16: Model representation and transformation components in the service creation component of the grid development tools with high level debugging components.

transforms this implementation back into its upper layer representation keeping the application specific logic, and re-generates all target platform specific binding code for another target platform.

A component creation module of the grid development tools should support two basic use cases: Interactive development of the service reflecting changes on one level in the other model layers by transformation, and automated use as a build tool acting on previously defined service sources.

4.3.3 Process Creation

The previously described components are geared towards a user group with a strong software development background, probably even a strong background in middleware or Grid development. However, rapid application development support should also enable developers with only limited expertise in distributed

application development to create applications for the Grid. Ideally, it should enable domain experts in various domains without a computer science background to construct applications as solutions to their domain specific problems that facilitate the available Grid resources.

BPEL has gained much attention and broad adoption for composition of component based business applications. The focus of BPEL is to enable the composition of basic web services into more complex processes. Its popularity in the business application domain makes BPEL very promising and interesting for process creation in the Grid domain, since many process execution, management and creation tools are expected to be developed in the future or are even currently under development. Another benefit of using the BPEL language is the ability to seamlessly integrate resulting Grid processes with other business processes in a corporate environment.

Two main considerations should drive the design of a Grid process editor supporting fast development of Grid application from high level perspective:

- It should provide the ability to adapt to the needs of different groups of developers, allowing Grid middleware experts to inspect and manipulate fine details of a Grid process (high-fidelity editing) while hiding complicated details from application domain experts (low-fidelity editing).

- A Grid process editor should foster collaboration among experts in a distributed environment. Ideally, it should support collaborative work on the process under development, building on the underlying communication infrastructure.

Since composition of the basic Grid services offered in the Grid middleware is the main focus of this grid development tools component, it should be more tightly integrated with certain management components such as the service discovery component.

The overall design of the grid development tools process editor is also based on a model driven approach. In this way the very popular model-view-controller (MVC) pattern [144] may be used to implement the user interface components of the grid development tools process editor. Use of the MVC pattern allows for the implementation of different view and controller components on a common model representation of the process. Figure 4.17 shows a conceptual overview of the core components of a collaborative Grid process editor. The design of the editor is separated into three layers, a presentation layer, a target system layer and the editor core. In section 4.2.7, possible grid specific extensions to BPEL were discussed. The process meta-model used to represent Grid processes in the grid development tools process editor should support this extended Grid process execution language combining a standard BPEL model enriched with the additional Grid specific constructs. Export to a concrete process execution engine or enactment environment is - similar to the support of different Grid

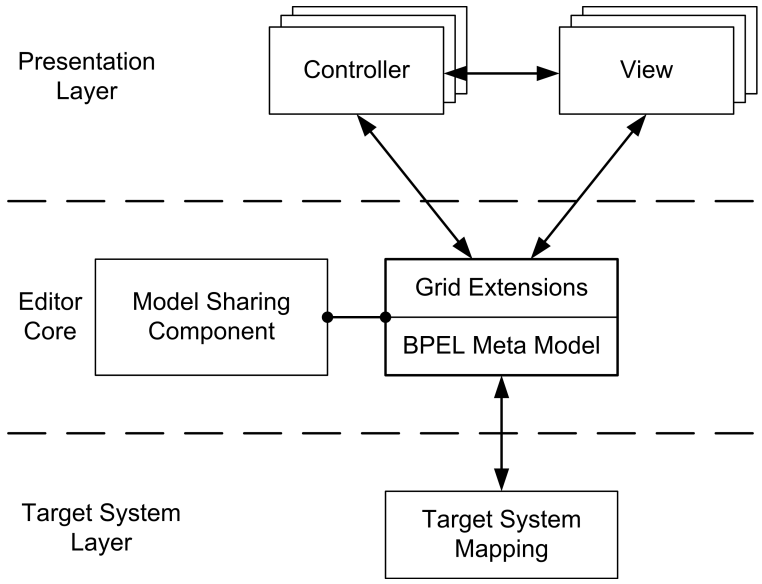


Figure 4.17: Component overview of a collaborative process editor for the Grid Development Tools.

target systems in the component creation module - handled by a target system mapping component. This target system mapping component implements the transformation from the BPEL oriented process model to a platform specific model representation and finally to source code for the process execution engine. It is possible to switch the target system layer implementation as well as the presentation layer implementation independently of each other, while keeping a constant implementation of the editor core. This allows to re-use different views and controllers for newly integrated process execution systems.

Independent implementation of the different layers is limited by internal dependencies of the implemented constructs in the core model used by the editor. There are two options supporting different levels of portability while still meeting the core design goals for the process editor (i.e. support collaboration and different level of detail views). As a first option a common core model may be specified and even standardized. Figure 4.18(a) shows this first approach which ensures that different target system mappings may be implemented independently from the presentation layer implementation and the presentation layer can be implemented indenpendently from any target system. Both layers are implemented against the fixed common core model. A second option (shown in figure 4.18(b)) gives up some of the independence between the target system mapping implementation and the presentation layer implementation. In this case, only the BPEL part of the core model is assumed to be a fixed model, while the Grid specific additions to the core model are chosen depending on the expressiveness of the core process model of a specific target system. The additional constructs in the core

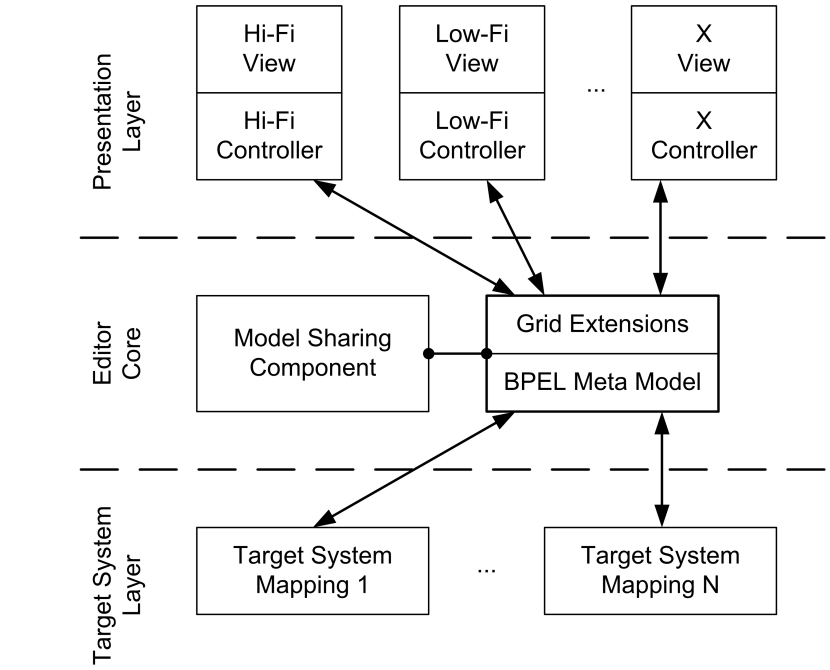
model need a corresponding implementation in the presentation components. In order to meet the goal of allowing both Grid experts and domain experts to use the editor on at least two levels of detail, the presentation layer components for the internal model elements must be implemented for both of these layers.

The field of process execution for the Grid is a rapidly evolving field. Currently, there is no standard for the expressive power of the core process models that everyone agreed upon. Research is going on in the area trying to find new process execution constructs that should be included in process execution systems for the Grid. This lack of a common standard and the greater flexibility in implementing new constructs lead to the choice of the second design option for the grid development tools process editor. Using this core design leads to the implementation of a process editor that already provides many constructs for the standard BPEL language. These constructs may be used by developers of concrete target systems when implementing an extension that supports additional Grid constructs found in their process execution environment. A sample implementation of such a process editor for the MAGE environment will be presented in section 5.3.2.

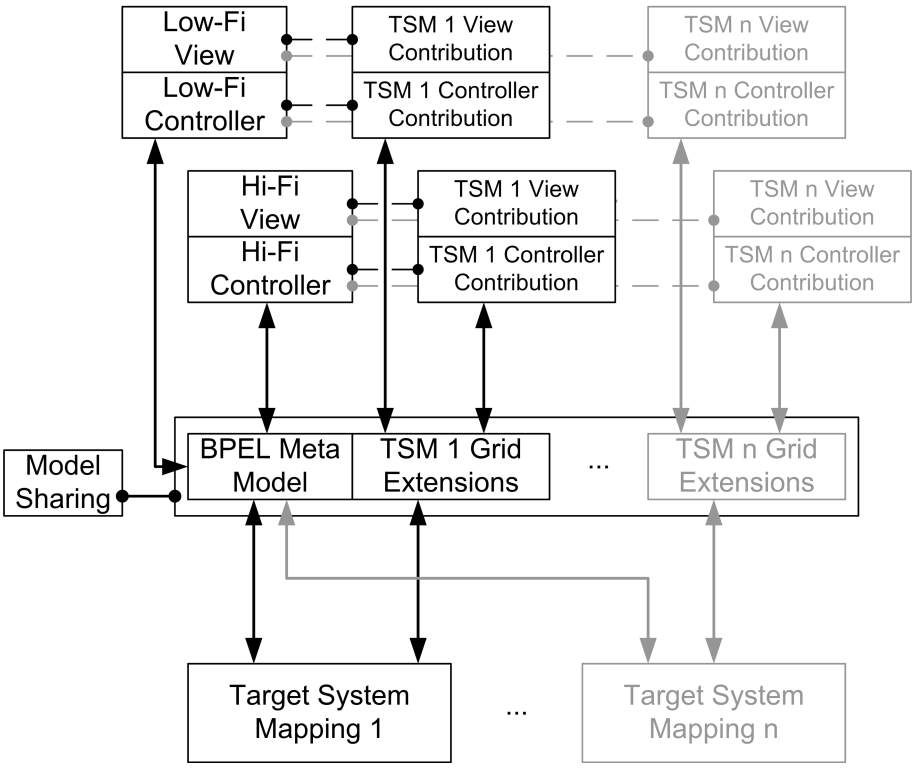
The collaborative nature of the process creation activity is the reason for the second central goal in the design of the grid development tools process editor: support for interactive collaboration. For this purpose, the editor core must implement a model sharing component. The purpose of this model sharing component is to allow different editors connected through a network to display and act on the same process model. The core design of the grid development tools process editor does not mandate a special implementation pattern for the model sharing component or a concrete synchronization or control strategy. Choice of implementation details in that respect is up to concrete implementation and available technology. Implementation of the necessary communication components using the underlying communication facilities offered by the target Grid system is encouraged since much of the management functionality and infrastructure of the Grid system can be reused, such as virtual organization management and access control.

The model sharing component as well as the different view components need to react to changes in the model. Therefore, they must be notified of changes to the core model. While the view component updates the display to relate changes to the user, the model sharing component propagates changes of the model to other editors currently acting on the same shared model. To allow the implementation of a distributed coordination protocol for the shared model, the model sharing component must also be able to lock the process model to prevent local modification when necessary.

An implementation of the grid development tools process editor should finally provide an interface to easily integrate various wizards and components automating complex modifications and application of default patterns into the editor. Every component of an implementation of a GDT process editor should



(a) Single standardized process meta-model.



(b) Fixed BPEL model with independent extensions.

Figure 4.18: Different realizations for the core process meta model.

follow a design approach that opens up the component for easy contribution from other components and integration with them. Such an addition may be a process pattern library that enables users to collect and use common patterns for Grid applications allowing them the fast creation of applications following a best practice approach with modifications for a concrete application. This component may query the core model for process elements selected by the user, strip the concrete binding to Grid services from the process structure and save this process model as a template for other applications in a pattern repository accessible by other users.

4.3.4 Interactive Debugging Support

The ability to discover errors in component implementations through interactive debugging greatly improves successful identification and correction of such implementation problems. An interactive debugger is usually part of most modern integrated development environments. For stand-alone applications, most IDEs allow the user to start an application under control of the debugger. The situation for large scale distributed applications is somewhat complicated by the distribution over different nodes in the network. Most integrated debuggers in modern IDEs nevertheless offer the ability to attach even to remote processes and enable developers to control the process execution and inspect in-memory value and execution states of the processes.

Many errors during the development of service-oriented Grid applications are related to the platform binding code or happen during invocation of the service within the middleware layer before the actual invocation reaches the user code or after the result is handed to the middleware. In such a case, it is desirable to not only control the custom application logic but to also have the ability to set breakpoints and inspect attribute values and execution states in the middleware itself. Meaningful interpretation of the actual execution state, on the other hand, requires access to the source code of the component under inspection. Many Grid middleware solutions are provided as an open source solution so that the source code is generally available and can be made available to the IDE's debugging component.

The assumption that exactly the same version of a component is deployed on each node involved in the current execution of a Grid application does, however, not hold for large scale Grid environments. Different versions of certain components may successfully be combined in one Grid environment if they still implement the same or interoperable interfaces since service oriented applications usually only rely on loose coupling of independent components. It is even more likely to find different implementations of a concrete component in an ad hoc Grid environment where different resource and solution providers join the network to combine their assets to a larger application environment.

In this scenario, the IDE integrated interactive debugger should be extended

by a Grid enabled support component. If a developer has the right to control a certain node for debugging purposes - a role that can be granted using an extension of the standard privilege control mechanisms for Grid environments - (s)he may use a special debugging service to attach to the Grid node, query for detailed version information and even retrieve the component source code for use in the integrated debugger. The debugging service client is then used to supply the interactive debugger with the right information that can then present the right source code for the component under inspection to the user.

The debugging service itself can be offered as a Grid service, and standard access control mechanisms for Grid services can be applied to the platform debugging service. Figure 4.16 shows the debugging support components in the lower part of the IDE. The integrated debugger is assisted by a client for the debugging support service in the remote Grid service container. This client is used to gather the specific information about the deployed components. Source code for the components can be retrieved from a module source code repository in order to make the source code available to the integrated debugger. Note, however, that the debug information service must be independent of the Grid service container under debugging since access to the service may otherwise be obstructed by the suspension of threads by the debugger.

An additional challenge for debugging of service-oriented Grid applications is their distributed nature. Applications are composed from many components distributed over the entire network. Successful debugging of such a distributed application potentially requires control over many different nodes and introspection of values and execution states. Therefore, the interactive debugging component of the GDT should make strong use of the node and service discovery components in the integrated Grid management module. As a contributor to the visual display of the Grid network, the debugger may present high level representation about the execution state of the different services and nodes in the graphical display of the Grid management component.

4.3.5 Grid Management Components

Grid middleware as a runtime system only provides architectural components to build Grid environments. Even though the central goal of the *ad hoc* Grid is to reduce the complexity of setting up and maintaining a Grid environment, some administrative tasks cannot be hidden from an administrator. Users may also require to keep control over some remaining tasks such as management of certificate authorities and assignment of access privileges. Another management aspect is the monitoring of the state of the Grid system. Management components for the ad hoc Grid need to hide complexity from their user while retaining intuitive control over configuration and maintenance aspects. This balance must be found to combine easy direct and manual management of a system as complex as a Grid environment with the ability to keep the ad hoc philosophy. Finally,

the previously introduced development and debugging components require access to certain management functionality or integration with the management applications to contribute their functionality to the management environment. As an example, the node and service discovery component may be accessed to search for Grid services. Those can then be used as component services during the construction of a new Grid process. Also, the interactive debugging component may directly be accessible in the node visualization, allowing the user to start debugging for a node with a single user interaction in the management user interface.

Most management functionality must be automated and built into the runtime system, in order to keep tedious set up and management tasks from the user. Those runtime components need to expose interfaces for management applications allowing them to directly retrieve information from the middleware and control the different configuration aspects of the core components. The interfaces to the core management components should be implemented as Grid service interfaces. Thereby, all middleware functionality such as discovery and access control applies also to the management interfaces of the middleware. The integrated management applications implement clients to the core management services. They should again expose their functionality to other components of the development and management environment. As a result, the management clients and development components of the GDT form a plug-in collection that expose APIs to directly access their functionality and extension points to extend certain user interface elements such as context menus with functionality contributed by other components in the GDT collection.

An additional aspect in the design of the management components of the GDT arises from the different intended user groups. While the development components are geared towards software developers, the management components are also intended to be used by administrators or even application end users. Therefore, the client side managment components and their user interfaces need to be integrated into end user management applications in addition to the integrated development environment. That means, all components shown in figure 4.12 must be usable in the integrated development environment, but also usable in a stand alone management application (represented by the gray box). This leads to the three layer separation in the management user module. The Grid service client is fully dependent on the implementation of the middleware module. Keeping the implementation of the client-side functionality of a management component independent of the user interface eases the re-use of the component in different user interface environments. Chapter 5.3.4 describes an implementation of the GDT client side management components that uses a single user interface implementation both for the Eclipse IDE and for stand alone graphical management applications yet provide an additional user interface implementation for the command line. The resulting high level design and integration of the client side management components with the underlying Grid middleware and

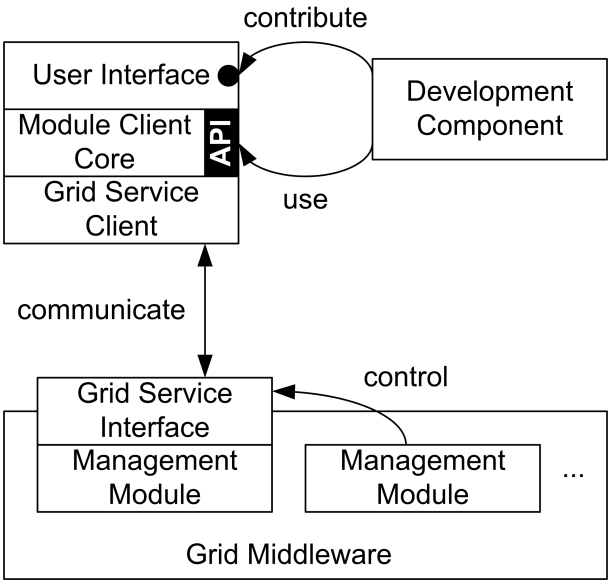


Figure 4.19: Relationship between Grid middleware, management and development components.

the development modules is shown in figure 4.19.

The client side management components of the GDT expose functionality directly corresponding to the functionality of the runtime modules. It can therefore be found in the respective design and implementation descriptions of the runtime components in sections 4.2 and 5.2. A description of the functionality is not repeated in this chapter for the individual client applications. In addition to the core module functionality, the management clients may provide additional functionality as either higher level compound functions in the management client core, exposing the functionality to upstream plug-ins, or as wizards automating complex or repetitive tasks on the user interface level of a management application. A discussion of these higher level capabilities is omitted, since it would exceed the scope of this work.

4.3.6 Extensibility

The previous description gives an overview of the design of a collection of loosely coupled components that together form the basis for a rich Grid development and management environment. A general principle to be used throughout any implementation of the previously described design is *extensibility*. This extensibility should be provided to enable additions to components on the same or a higher conceptual level. Every component of the Grid development and management environment should therefore provide access to its functionality through an API that can be used by other components realizing a higher level functionality. An

example for such components are user interface wizards that provide more complex functions to the user, based on the functionality of other components such as the service generation or process creation components. To allow extension of the internal functionality of a component, it should offer well defined extension points.

4.4 Summary

In this chapter, the design of a service-oriented ad hoc Grid middleware solution was introduced. An implementation of an ad hoc Grid solution based on this design - The *Marburg ad hoc Grid Environment* (MAGE) will be described in the next chapter.

The MAGE system is designed as a combination of a runtime execution environment - the actual Grid middleware supporting service-oriented Grid applications - and a management and development environment that supports application developers, application providers and end users in setting up and maintaining their ad hoc Grid environment.

The ad hoc Grid middleware uses a modular design approach. Grid service discovery and communication is based on an exchangeable P2P infrastructure. A fundamental feature of the middleware design is the inclusion of Grid service deployment functionality as a service, that makes component deployment in the Grid a function usable in higher level applications.

This ability to introduce foreign code into a Grid service container running on a resource providers' node leads to security implications that are dealt with by a Grid service isolation approach. Since many Grid applications rely on legacy components developed over a long time, the isolation scheme included in the design of the ad hoc Grid middleware also addresses the issue of isolating these native components. As a last runtime component, a component for Grid process execution and a corresponding extension of the BPEL language were described to support the creation of higher level applications by composition of basic Grid services. The design of a data handling mechanism was also introduced, forming the basis for the development of Grid processes since it allows to more easily transfer large amounts of binary data in a service-oriented Grid environment.

In the second part of this chapter, the design of a set of Grid development tools was introduced that help to reduce the complexity of Grid application development. Grid developers are supported to easily construct Grid services with a model driven approach, collaboratively develop complex Grid applications from these component services and interactively debug the resulting components and applications.

5

Implementation

5.1 Introduction

In this chapter, a prototypical implementation of a service-oriented ad hoc Grid is presented, dealing with each of the requirements mentioned in section 2.3.

The basis of the implementation described in this chapter is the Globus Toolkit 4.0 (GT4) deployed in Tomcat 5 [20], since it is the most widely used implementation of the Open Grid Services Infrastructure. Furthermore, since GT4 is deployed as an Axis Service in Tomcat, it offers a number of access points into the system via configuration of the Axis service, allowing easy integration of new features.

For the Grid development and management components, the Eclipse platform as an extensible integrated development environment was selected. A second key benefit of using the Eclipse platform is the ability to reuse Eclipse plug-ins not only in the integrated development environment but also in stand-alone applications - Rich Client Platform applications - geared towards end users that are not developers.

The rest of this chapter is organized similar to the structure of the previous design chapter. Therefore, this chapter is also separated into two parts focusing on the runtime components of the MAGE ad hoc Grid middleware in the first part and the Grid Development Tools for Eclipse in the second part. Every part focuses on the implementation of a component of the MAGE middleware following the design in the previous chapter. Several implementation issues for the components presented in this chapter have already been published in [77, 81, 80, 78, 79, 97, 118, 83, 156, 155, 153, 154, 157, 82, 158].

5.2 Runtime Components

5.2.1 Peer-to-Peer Infrastructure

An introduction to the nature of possible Peer-to-Peer information infrastructures was given in section 4.2.1. For a first prototypical implementation of MAGE, the Resource Management Framework (RMF) [77, 146], a P2P infrastructure developed by Siemens AG, Corporate Research, Munich, Germany was employed. The purpose of the RMF is the creation of a flat information space for the storage and retrieval of XML documents - referred to as resources in the RMF. The peers automatically form a network overlay structure that allows them to be addressed using a generalized addressing scheme. The RMF API abstracts from any details of the underlying network as well as the concrete overlay routing mechanisms used (that can be Chord, Tapestry or Napster-like) allowing a resource centric view on the information space.

The basic operations supported by the RMF are the publishing of a resource, retrieval of a resource using a given resource ID, synchronous as well as asynchronous search for resources and subscription for reception of change notification. Such notifications are generated by the system upon certain events such as the publication of a resource, its deletion from the network or changes occurring on the resource. Search and subscription operations are based on XPath queries into the contents of a resource.

The RMF handles storage of the resources on individual nodes as well as replication of resources in order to make the information space resilient against failure of individual nodes. Storage of a resource is based on resource leases. If the lease for a piece of information is not renewed by the owner of the resource within its expiration time, the resource is purged from the information space.

The basic element managed in the information space of the RMF is a resource. The structure of this XML element may be arbitrarily defined by an application. In order to allow the RMF resource registrar to manage this resource in the information space of the RMF, a number of child elements from the RMF registrar name space should be present as child nodes of the root element of the resource. The most important elements of the registrar name space are: (a) a globally unique identifier for the resource; the use of UUIDs is suggested in order to prevent ID collisions throughout the entire information space; (b) a user friendly name for the resource; this name may be used in applications for the representation of the resource regardless of the application specific content of the resource document; (c) a list of keywords that are used to identify regions of the information space where searches are conducted.

The unique ID of the resource is the only mandatory element of any resource to be published. It is possible to take virtually any XML document, include an ID as a child element of the document root and then publish the document in the RMF as a resource.

For a second implementation of the discovery and communication components of MAGE, FreePastry [76] was used. FreePastry offers a P2P routing infrastructure that forms a flat address space similar to the one provided by the RMF. The main feature used in the second prototypical implementation of the discovery and communication component of MAGE is a reliable application layer multicast layer called Scribe [37] that is built on top of FreePastry. Scribe automatically organizes nodes in a hierarchical multicast tree that guarantees transmission of a message to all nodes in the tree. The basic FreePastry routing layer is used to quickly discovery individual node failure and repair the multicast tree.

5.2.2 Node and Service Discovery

To enable automatic service deployment in an ad hoc Grid environment, the participating nodes must be discovered first. Due to the potentially large size of future Grids, manual discovery as practiced in existing Grid environments is not an option. Since the Grid can cross the boundaries of organizations, a simple multicast or broadcast will not reach all potential participants without proper preconfiguration of the network infrastructure which is unacceptable in an ad hoc Grid environment. An automatic discovery mechanism is needed to find nodes willing to participate in the Grid. A central registry system is easy to install but does not scale well and introduces a single point of failure. For ad hoc Grids, a decentralized discovery mechanism is vital to cope with the fluctuating topology and large number of participants.

The peer-to-peer community has spent significant effort to solve the node discovery problem in large, heterogeneous and unreliable networks. Peer-to-peer and Grid computing systems have a number of similarities. Both systems aim to bring together distributed resources. In general, peer-to-peer systems are designed to fulfil a single task (e.g. file sharing), while Grids are multi-purpose and offer greater flexibility for distributed application design. The advantage of peer-to-peer systems is that they are easier to install, configure and administer. Typically, there is no central coordination needed at all. Current Grid systems are relatively small encompassing several thousands of nodes, while peer-to-peer systems can connect millions of nodes [42, 92] using only the limited resources of personal computers. Iamnitchi and Foster [99] present a more detailed comparison of the two technologies. The authors also state that peer-to-peer applications are becoming more complex, offering general distributed computing capacities. At the same time, Grid systems are growing bigger and thus the differences between the two paradigms are likely to disappear over time. Although a number of papers [99, 100, 124, 168] discuss the benefits offered by the confluence of peer-to-peer and Grid computing, most of the projects suggest to integrate peer-to-peer computing ideas only for particular aspects of Grid computing rather than integrating a fully fledged peer-to-peer solution at its core.

The first implementation of node discovery for MAGE is based on the RMF

```
<node>
 <groupName>MarburgAdHocGroup</groupName>
 <capabilities>
  <os>
   <name>Linux</name>
   <version>Debian 3.1</version>
  </os>
  <cpu>i686</cpu>
  <memory>1024</memory>
  <peripherals>
   ...
  </peripherals>
  ...
 </capabilities>
</node>
```

Figure 5.1: A node capability record.

described above and is aimed at providing group oriented discovery mechanisms. We assume that each of the different participants in the ad hoc Grid decides to be part of one or more collaboration groups in the system. A node registers a capability record under the group name in the RMF information space. This capability record contains static information about the node such as the operating system, processor type, total amount of installed memory and special resources available at the node (such as special sensor equipment) (see Figure 5.1).

An application requiring service deployment to other nodes can then perform a search operation in the information space, using XPath expressions to further constrain the returned node information records. As an example, an application can easily constrain the search for nodes running Linux as an operating system with the following query:

```
/node/capabilities/os[name='Linux']
```

The static property schema has been defined as an XML schema, a mapping to the RMF information space was automatically generated from this XML schema specification using the approach described in more detail in [80]. The binding definition is basically a combination of a subset of the GLUE [15] schema and custom deployment related information (pre-installed software and libraries). A P2P adapter between RMF and the GT4 MDS service has been implemented which transforms information stored in the MDS into RMF resources and publishes them. After gathering static information about a node from the information space, an application can directly query the node for its current state, such as the available memory, its computational load and policies for the deployment of services.

Service discovery in the RMF based discovery component of MAGE is implemented in a similar fashion. A custom P2P resource binding has been defined that maps service name, port type name and endpoint addresses of a WSDL description of a service into the RMF information space. This approach is not limited to an ad hoc Grid environment. More details about the design and implementation of this approach and its application in a web service based business application context can be found in [79].

A second implementation of the MAGE discovery component uses a direct querying approach based on FreePastry and Scribe. MAGE nodes using FreePastry P2P Discovery join the network and form an overlay network. Every node can join any number of Grid groups, each group is represented by a Scribe multicast tree. To query the group, a client can send a complex XPath query over the GLUE compliant information in the local MDS information resource of every node through the Scribe tree. Results are propagated and aggregated along the inverse path of the query. Service discovery is realized similarly in the FreePastry based discovery implementation.

5.2.3 Service Communication

Message Handling in the Globus Toolkit

The Globus Toolkit uses the AXIS web service engine developed by the Apache Jakarta project to handle Grid service calls. AXIS is implemented as a web application that can be hosted either in a custom container distributed with the GT4 release or inside a Tomcat web application container. MAGE uses Tomcat as the web application container to host the Globus web application.

The web application container handles TCP connections and decodes HTTP messages into header and body information. These request objects are then passed to the AxisServlet that acts as the entry point to the AXIS web service engine. The AXIS engine constructs a message context for every invocation that is processed by so called handler chains. Handler chains are easily configurable ordered lists of handler implementations that transform the message context. GT4 uses the WS-Addressing standard for identification of the target service to invoke; an addressing handler is configured for the request handling chain to interpret the WS-Addressing SOAP message headers and set an attribute that identifies the target service for creation of the appropriate Java classes by another handler in the request processing chain.

Tunnelling SOAP Messages through the P2P Network

There are a number of possible points for message interception in the client and message injection in the service container. Tunnel endpoints at both ends can act on the SOAP messages before they are encoded as HTTP messages, on the HTTP

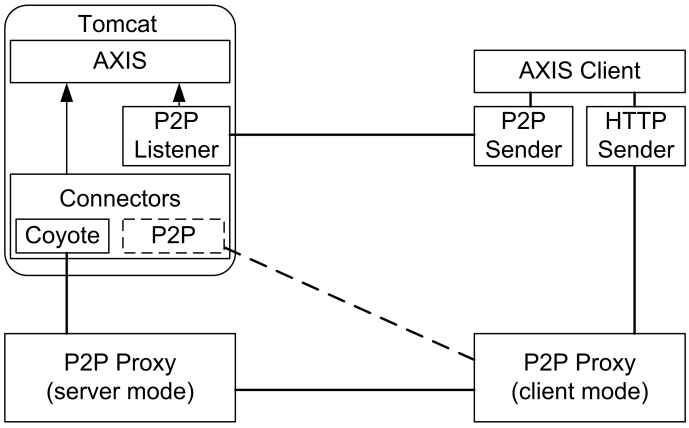


Figure 5.2: Interception and injection of SOAP messages can happen at different points in the infrastructure.

messages that have been constructed by the web service hosting environment or by a transparent intermediary element that can be configured as a proxy for the client. In the latter case, the HTTP request is relayed to a receiver running at the target host that acts like a regular HTTP client and issues a request to the service container. In both other cases, a component is integrated into the regular message handling infrastructure that injects the message containing the call. Combinations of the interception and injection points are also possible. For example, the HTTP message could be intercepted by a proxy implementation and then directly injected by a custom implementation of a protocol connector in the hosting environment. The different interception and injection points are depicted in Figure 5.2. The top-most connection was implemented using a custom P2P sender on the client side as well as a P2P listener that directly invokes the AXIS engine inside Tomcat.

A vital requirement of using such a message tunneling infrastructure is the identification of the target host. Without any changes to the client or the server, the original target endpoint address for a service is the only indication for the target service. A proxy can use the previously described service discovery component to search for a service that has been registered to use the target endpoint URL in the intercepted method. A problem of this approach is the ambiguity of the endpoint address. The Globus toolkit generates the endpoint address from the IP address of the local host. Different private networks behind NAT routers can share the same private IP address space (e.g addresses of the form 192.168.xxx.yyy). Even consideration of the service name or the service port type is likely to fail in the ad hoc Grid scenario since certain worker services are expected to be deployed on many nodes exposing the same service interface under the same IP address where the actual target nodes differ.

To circumvent this problem, the implementation follows an integrated ap-

```
<responseFlow>
 ...
 <handler type="java:de.fb12.WSDLWeaveRMFHandler" />
</responseFlow>
```

Figure 5.3: Handler chain configuration.

proach of early message interception and late injection, keeping the message handling chain as short as possible, which also limits the amount of time needed for message encoding and decoding. A preliminary step to using this streamlined implementation is the inclusion of a custom reply handler in the global reply handler chain. Figure 5.3 shows the changes that can be applied to the configuration of either the global or transport specific response handling chain.

The invoke-method of the class WSDLWeaveRMFHandler gets invoked by the engine, whenever a WSDL description of a service is generated. This method takes the WSDL document from the message context and adds a custom endpoint URL of the form rmf://peerID/wsrf/serviceName for the service. The client can use this target endpoint address instead of the HTTP endpoint later on.

For the actual service invocation, a listener thread is started for the Globus engine. This thread handles incoming connections from the P2P network. The message transmitted using the P2P protocol is decoded similar to the message handling performed by the HTTP connector, and handed over to the AXIS engine that handles the actual invocation of the target service instance. The reply message is then transmitted to the client using the P2P communication layer. The underlying P2P network supports TCP like connections over the multi-hop overlay network. Message injection in the platform can be configured to use access methods and content encryption and signing facilities already present in the Globus toolkit. In addition, end-to-end encryption of network connections between client and server is supported.

Support for the P2P communication infrastructure can be configured into the existing Globus infrastructure by referencing custom URL- and protocol handlers that MAGE implements. Registration of the custom P2P transport protocol can be achieved similar to the custom security protocol registration already performed by the standard Globus toolkit. The MAGE toolkit allows the use of P2P communication by selecting P2P endpoint addresses without the need to ever touch any P2P specific code.

Since FreePastry does not provide a connection-oriented and reliable communication mechanism, a reliable transport layer has been implemented for FreePastry to allow tunneling of service invocation messages through both P2P networks. The MAGE system automatically registers custom Java URL handlers to allow users and applications the specification of endpoint URLs using protocol identifiers for both P2P infrastructures (i.e. `rmf://...` and `pastry://...` URLs).

5.2.4 Service Deployment and Administration

The deployment of a service currently requires a Grid service archive (GAR file) containing the needed classes, schema files and deployment descriptors that make up a service. Users of GT4 are supplied with Ant tasks that handle the distribution of the contents of this GAR file into the local standalone GT4 environment. The Ant tasks extract and copy the jar files containing the class files of the service into the local web application directory. Schema files are copied into the schema repository. The current deployment strategy of GT4 requires the restart of the entire WSRF web application, thereby killing every other Grid service currently running. Furthermore, direct access to the machine running the GT4 application is required because the Ant tasks perform all copy operations locally.

Neither the first nor the second property of the deployment mechanism is feasible for an ad hoc Grid environment with a frequently changing collection of nodes. In this environment, an application has to make sure - through dynamic service deployment - that the required service is present on every node it wishes to incorporate into its application flow.

To enable this, the Axis web service engine utilized by GT4 was modified to allow dynamic loading and unloading of Grid services. The MAGE hot deployment service (HDS) provides applications with the capability to remotely deploy, undeploy and redeploy services onto a running node.

The basic steps the HDS needs to perform to deploy a service are:

- Register the service description with the AXIS/WSRF request handlers.
- Register the service naming description with the JNDI registry.
- Make the schema files available to the WSRF environment.
- Make the service class files available to the class loader.

Currently, the need to load additional classes and dynamically replace them was not anticipated or governed by the WSRF specification or the Globus Toolkit 4 implementation. Hot deployment is a central feature of the ad hoc Grid environment. An implementation of the HDS for the Globus Toolkit 4 is presented in the following. To enable this functionality, a class loading mechanism is introduced into the GT4 Grid middleware.

Grid services in GT4 are separated into three classes: The service resource class, a service home class and the service implementation itself. The service home class is used to load resources attached to a service and the service classes themselves. MAGE provides the class `HotResourceHomeImpl` as implementation of the `ResourceHome` interface in order to leverage a custom class loading mechanism into GT4. The `ResourceHome` is responsible for creating the `ClassLoader` hierarchy which will be used to load the service classes. It distinguishes between different instances of the class loaders by acquiring the service context

```
public void setResourceClass(String clazz)
    throws ClassNotFoundException {
   String serviceName =
     AxisEngine.getCurrentMessageContext().getTargetService();
   String basePath =
     ContainerConfig.getConfig().getInternalWebRoot();
   String relPath = basePath+servicePath+libPath;
   ClassLoader cl =
     JarClassLoaderManager.createLoader(serviceName, relPath)
   resourceClass = cl.loadClass(clazz);
}
```

Figure 5.4: The setResourceClass operation in HotResourceHomeImpl.

from the AXIS engine inside the GT4 web application. It also registers all class loaders created by itself at a central `DisposableClassLoader-Manager` and the Axis `ClassUtils` class loader cache, so they can be accessed later during undeployment. The code snippet in Figure 5.4 shows the main operation of the `HotResource-HomeImpl`. First, the service name is extracted from the current message context. Then, the path where the jar files of the service are stored is generated based on the container configuration and an arbitrary path extension. In case of MAGE the `basePath/WEB-INF/lib/serviceName/` was chosen. Based on that, a `JarClassLoader` was created that is capable of loading all classes contained in all jar files in that directory. The `JarClassLoaderManager` also informs the Axis `ClassUtils` that it is now responsible for this service.

This is a non-intrusive way to introduce a custom class loading mechanism into GT4, since the `ResourceHome` implementation can be specified for each individual service. A service wishing to be hot deployable merely must use the `HotResourceHomeImpl` instead of the standard `ResourceHomeImpl`. This is the only change required to make a service hot deployable and reloadable. Hot deployable and standard services can be run side by side by using the different `ResourceHome` implementations. Figure 5.5 shows the relationship of the `ResourceHome` implementations and class loaders.

The process of loading a service class is as follows. When a service is first requested, the org.globus.wsrf.jndi.BasicBeanFactory loads the `HotResourceHomeImpl` class in the standard Axis `WebAppsClassLoader`. The `HotResourceHomeImpl` is responsible for creating the disposable class loaders which will later load the service classes and the attached resources. When the `setResourceClass` method is called by the `BasicBeanFactory`, the `CurrentMessageContext` from the Axis engine is parsed to discover on behalf of which service the method is being called, thus allowing the framework to create one and only one `ClassLoader` for each service. The MAGE `JarClassLoaderManager` and the modified Axis `ClassUtils` are informed of

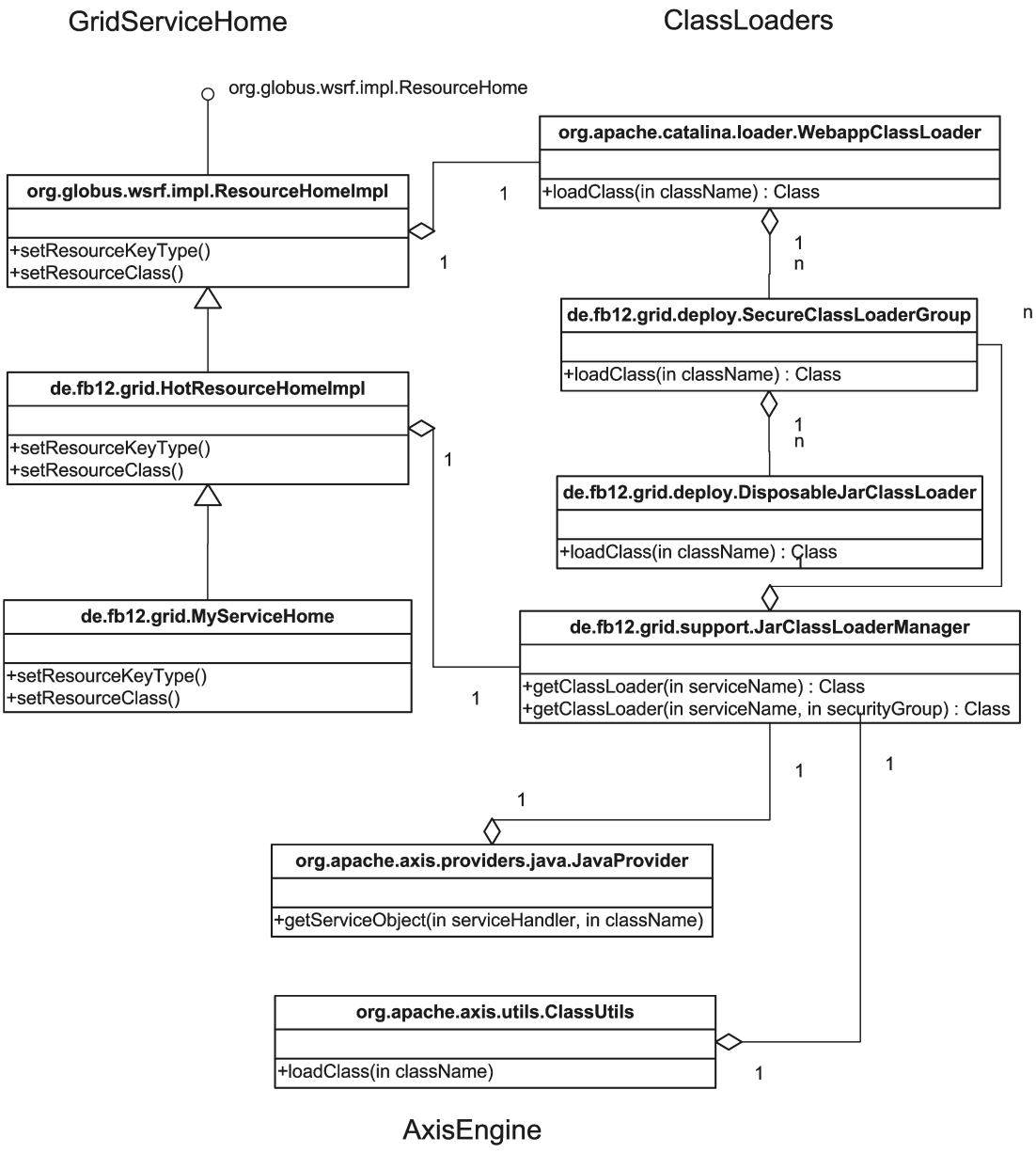


Figure 5.5: Relationship of the ResourceHome implementations and class loaders.

the service to `ClassLoader` mapping. Once the `ResourceHome` is in place, the `BasicBeanFactory` informs the home object of which class is the main service class. As mentioned above, the `HotResourceHomeImpl` attaches a disposable `ClassLoader` to the service. `Class` and `ClassLoader` are then used by the org.globus.axis.providers.RPCProvider to instantiate the actual service object. The `HotResourceHomeImpl` makes sure that the class is loaded by the proper `ClassLoader`. Then, everything is in place and the service can be accessed via the `JavaProvider`.

To remove a service, the in-memory registry entries are deleted from the JNDI and Axis system registries. Once the service information has been removed, no new service instance can be created. Running instances of a service previously created are untouched by this process. To deploy a new version of a service, no explicit unloading of the old service classes is required, since the new version of the service will be created using a new ClassLoader. If, in addition to the service information, the service instances are to be removed, the central manager used by the JarClassLoaderManager can be used to access the ClassLoaders of the separate services to free the resources and unload the classes. Only in this case the jar files can be deleted, since otherwise active services might try to lazy load classes after the containing jar files have already been removed.

Figure 5.6 shows a snapshot of the ClassLoader hierarchy in the system, after three different Grid services were instantiated. The GT4 environment and the MAGE hot service deployment mechanism are loaded by the Tomcat WebAppClassLoader. The remote clients call the deployment service where the HotFactoryCallBackImpl creates a DisposableClassLoader for each service creation request received. The GridServiceHandle is then returned to the clients. After the instantiation of the third service 'C', its implementation was updated and reinstantiated as 'C*'.

This central component of the ad-hoc Grid now allows services to be installed on-demand on nodes running the HDS.

5.2.5 Service Security

In this section, the security threats of an ad hoc Grid environment [155] are presented in more detail. Since the separate Grid services running on one Grid node are all hosted by Axis in GT4, they run within the same JVM and the classes are loaded by the same class loader. As a consequence, interaction between the classes is possible, thus offering malicious code the possibility to harm running services.

A first attack vector on the data of another Grid service instance are static fields and references and singleton instances. A malicious service could simply use the same package as the service under attack and then access the static members of the target service. The usual way to protect access to the classes in a standard Grid environment is to use a security manager for the JVM execution the Grid

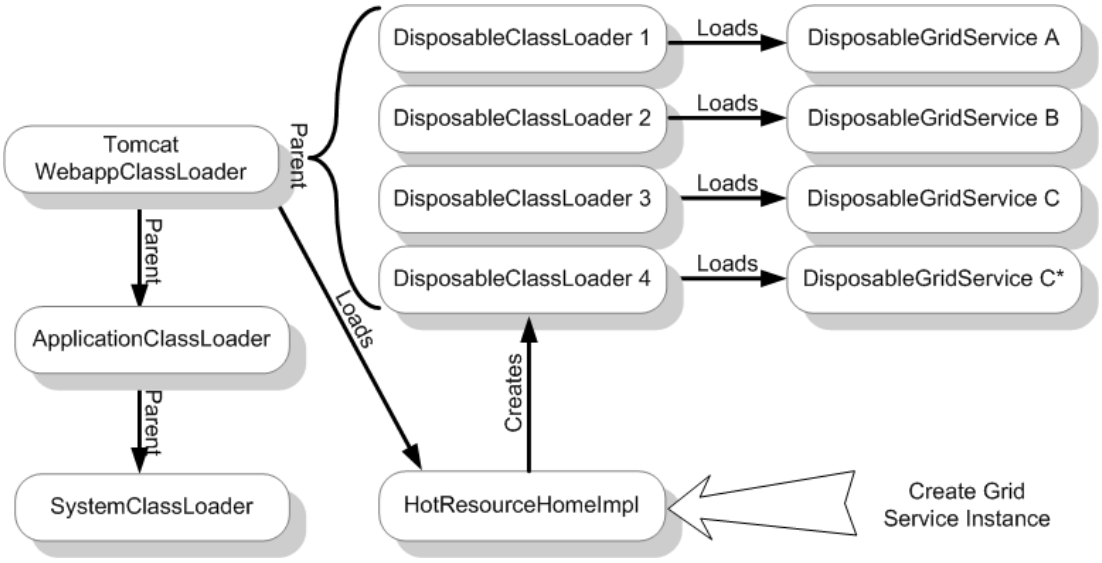


Figure 5.6: Class loader hierarchy.

service container. This security manager can be configured to restrict access to the packages of the service implementation, preventing the creation of new classes in the protected packages. This approach is, however, not feasible in an ad hoc Grid environment since the security manager must be configured before the start of the JVM. It would also hinder legitimate deployment of additional services or components in the same packages.

A second attack scenario uses preempted class definitions. If a dynamically deployed service needs to be loaded, a malicious service could re-define a component class of the Grid service under attack, i.e. define a malicious class with the same fully qualified class name. If the attacker forces loading of the class prior to the legitimate classe, the JVM will prevent re-definition of the class and use the malicious component instead of the legitimate one. Most standard protection mechanisms against these attacks can be circumvented by e.g. explicitly loading the malicious classes from the malicious code.

The MAGE platform needs an isolation mechanism for Grid service isolation that does not have the aforementioned problems. As a solution to the above security problems, a class loader based sandboxing scheme has been implemented which protects the services from illegal access.

An Approach to Intra-Engine Service Security

In GT4, the intra-engine service attacks described previously are made possible by the fact that GT4 loads all Grid services within the same class loader. The basic idea of the MAGE solution to this problem is to use the ClassLoader hierarchy introduced in the previous section to enable the dynamic loading and reloading

of classes to also provide intra-engine service security. In its most basic form, each Grid service is loaded within its own ClassLoader functioning as a sandbox and as such its classes and resources are private and cannot be accessed by any other service. This ensures that services using singletons and static references or static class members cannot be attacked by malicious services, this approach also prevents injection of foreign code into the service implementation by preempted loading of malicious code.

Service Sandboxing in Axis

To enable the required secure loading process, the `ClassUtils` and `JavaProvider` classes provided by Axis needed to be modified. To ensure that service classes are only loaded by sandboxed class loaders, the `ClassUtils` were modified to retrieve the current message context and check whether the current loader request was triggered by a service or by the container itself. If the load operation was triggered by a service, it checks whether the service is registered with the MAGE `ClassLoaderManager` and it should be protected. If that is the case, the class is loaded using the appropriate service `ClassLoader` and Axis is prevented from loading the classes in its `WebAppsClassLoader` and thus breaking the sandbox. Otherwise, the Axis class loading is unmodified, allowing all normal operations to proceed unhindered. The second place where Axis could try and load the service classes into its own `WebAppsClassLoader` is the `JavaProvider`. Similar to above, MAGE checks whether the service is registered with the ad hoc Grid framework and if that is the case the Axis loading mechanism is preempted and MAGE's own class loaders are used.

Inter-service communication is limited to using web service calls by these modifications to the internal class loader structures used by AXIS. Therefore, authentication between services using the standard Grid service access control mechanisms is enforced.

Secure Sandbox Groups

If it is necessary that two services are able to communicate directly using class references to create a composed web service application, they must group their Class-Loaders together using a SecureGroupClassLoader provided as part of the MAGE intra-engine service security infrastructure. A service specifies which group it wants to join either by passing the groupId to the Hot Deployment Service or by setting the parameter `<parameter name="group" value="groupId"/>` in the serverdeploy WSDD of the service. The SecureGroupClassLoader responsible for the group is a parent ClassLoader to all service ClassLoaders in that group. It enables inter-service communication in two ways: First, separate communication classes are placed in the SecureGroup-ClassLoader which can be accessed by all child ClassLoaders. This is the preferred way as defined by Java to allow classes

in sister ClassLoaders to communicate. For instance, the interface class of an object to be used by classes in two sister ClassLoaders is placed with the parent so it can be accessed by both children. The implementing classes are placed in both child ClassLoaders and object references can be passed between ClassLoaders as long as only the interface defined in the parent ClassLoader is used. This is the traditional way to allow code-based interaction between services but it requires that the communication classes are placed in the parent ClassLoader. The disadvantage of this approach is that if the communication classes need to be replaced, all child ClassLoaders must be discarded because the SecureGroupClassLoader must be replaced. So, even if only two services use the communication classes, all services must be undeployed to update the communication classes. To avoid this problem, the SecureGroupClassLoader is capable of emulating a flat namespace for its child ClassLoaders while still allowing hot deployment and hot undeployment of component parts of the composed web service application.

When a service ClassLoader joins the SecureGroupClassLoader, the SecureGroupClassLoader checks which classes the service ClassLoader is capable of loading and stores that information internally. If a different service within the same group tries to load one of those classes, its own ClassLoader will not be able to find the class and thus asks its parent, the SecureGroupClassLoader. The SecureGroupClassLoader then checks whether one of the other service ClassLoaders can load the requested Class and passes the request on to that ClassLoader before passing the request on to its parent, the WebAppsClassLoader. This only works if each Class is only defined once within all ClassLoaders in the same group. If different versions of one class can be accessed from the same ClassLoader, TargetInvocation and ClassCastExceptions will be the result. However, this ClassLoading mechanism was designed to allow tightly coupled web services to be composed into a web application, so it is very unlikely and undesirable that the same class will be defined in two different places, since the idea of tight integration was to be able to reuse the classes of the other services. In the case of such class duplication, the less tightly coupled composition via service calls is the preferred way of linking different web services, and the grouping function should not be used.

Undeployment of Grouped Services

Undeployment of services is more complex if the service to be undeployed is in a group, since classes from services loaded in different ClassLoaders can have references to each other. To prevent these classes from being undeployed when one of the other services tries to access undeployed classes, the SecureGroupClassLoader stores the information which service ClassLoaders have interacted with each other and denies undeployment requests to these services unless all other connected ClassLoaders are undeployed. Services loaded in the same group but which have not accessed classes of the services to be undeployed remain unaf-

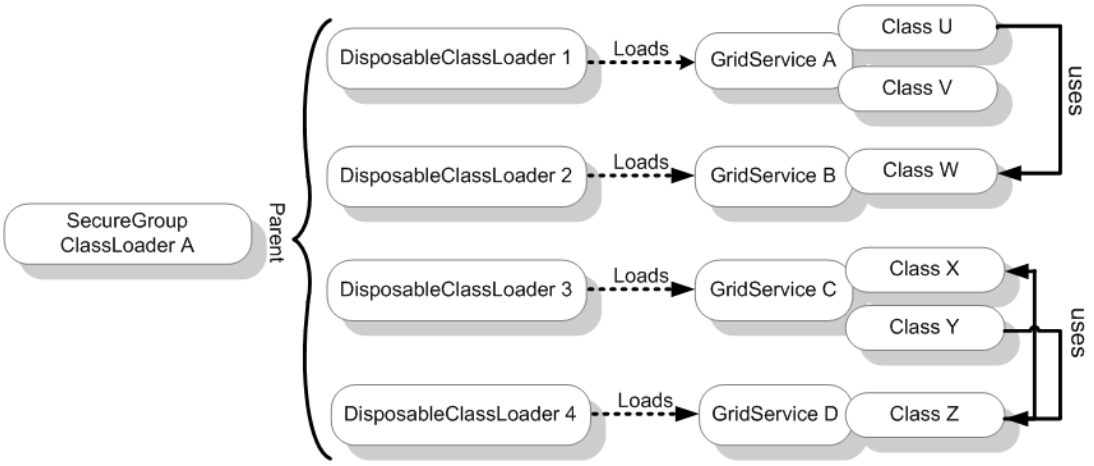


Figure 5.7: ClassLoader group interaction.

fected by this process. This is a clear benefit compared to the standard approach of placing the communication classes in the parent ClassLoader.

As an example, Figure 5.7 shows four Grid services which are joined into a group by one SecureGroupClassLoader. Service A defines classes U and V where U uses W which is defined by Service B. Service C defines classes X and Y and Service D defines class Z. Class Y uses Z and Z uses X. That means, Service B cannot be undeployed while A is alive, and Services C and D can only be undeployed together.

Group Access

To be able to securely group different service ClassLoaders together, access control to the grouping function is required. Currently, the MAGE implementation regards the holder of a public/private key pair to be the owner of a service group. The private key is used to sign Grid Service Archives which are to be deployed into the group. The public key is used to identify the group and to check whether the GARs submitted for deployment are permitted to join the group. When the deploy method in the HotDeploymentService is called, the HotDeploymentService checks whether the GAR submitted for deployment was signed using the private key for that group. If the GAR was signed correctly, the deployment process is allowed and the service ClassLoader is added to the SecureGroupClassLoader of that group; if not, the deployment process is aborted and no changes to the Grid environment are made.

Figure 5.8 shows a snapshot of the complete ClassLoader hierarchy in the system, after four Grid services have been instantiated in two separate groups. Services A through C are deployed in the same group and can access each others' Class definitions. Service D is deployed in its own group and thus is protected from direct code access by any of the other services deployed on this node.

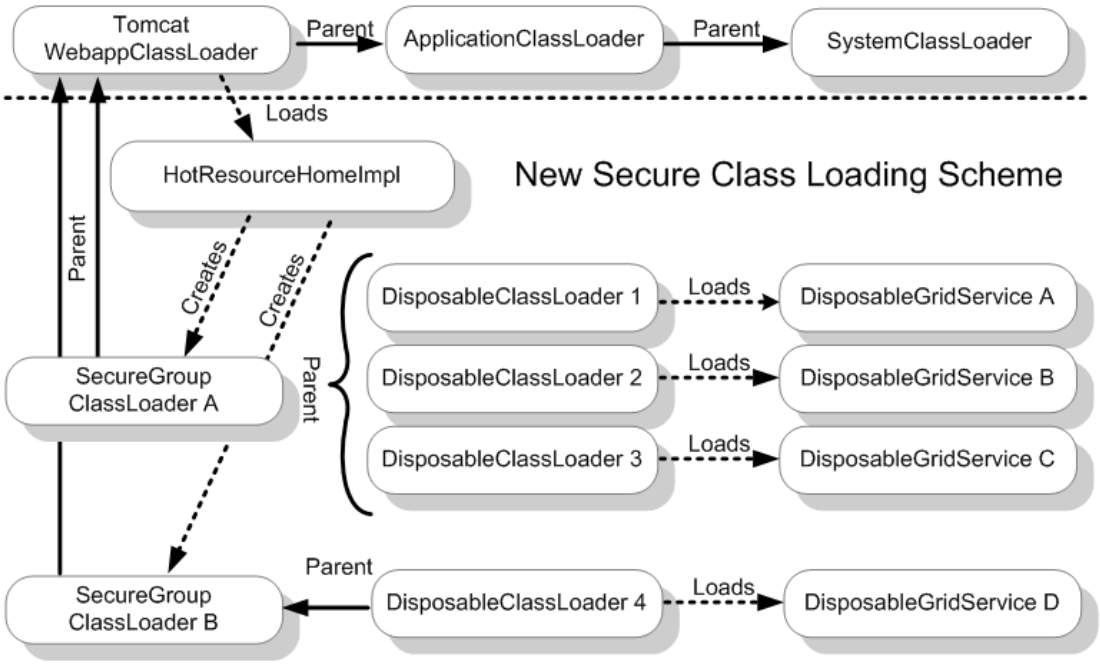


Figure 5.8: Hierarchy of the ClassLoader instances.

Isolation of Legacy Code

While a strengthened sandboxing of pure Java code running within the same Java virtual machine can be realized with the previously presented approach, the common occurrence of native code (e.g. C/C++, Fortran) in scientific Grid applications leads to a number of security issues which are not handled by a pure Java security mechanism. There are two possible ways to integrate native code into Java Grid services: by spawning a child process of the JVM with the java process builder that executes an external command or application, or by directly embedding native functions in the Java based service implementation using the Java Native Interface (JNI) [164].

In an environment where multiple services are hosted within the same Java virtual machine, security threats from native code arise from the fact that child processes are usually created with the user rights of the parent process, i.e. under the user ID of the user that started the service hosting container. For MAGE the use of process separation and confinement into secure sandboxes for native or legacy components is proposed, in order to allow a flexible and fine grained definition of execution policies in an open multiple service environment. The MAGE implementation does not require multiple instantiations of the JVM for isolation of different services, it is also independent of the JVM implementation allowing any JVM to be used to run the Grid or web service hosting environment.

The JNI interface is organized like a C++ virtual function table. It is passed

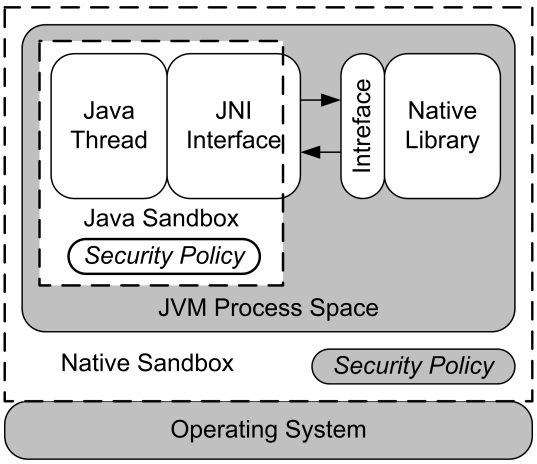


Figure 5.9: Standard access to native code through the JNI interface.

by reference to the native implementation and managed by the JVM per thread (i.e. a native method may be invoked from different threads and therefore receive different JNI interface pointers, invocations from the same thread are guaranteed to pass along the same pointer). The structure itself contains a reference to an array of function pointers to implementations of the JNI interface methods. Besides passing an invocation result with `return`, the native method must use those JNI interface functions for access to any method or field in Java classes and objects managed by the JVM. The native methods are compiled into shared libraries and Java code using native implementations loads those shared libraries using `System.loadLibrary`. Native code is then executed in the process space of the JVM which leads to the serious threats described before. The native code cannot be further constrained on a fine grained per-service level, only confinement based on the JVM process owner is possible. Figure 5.9 shows the relationship and confinement area using a standard approach for interfacing with the native code through the JNI.

As a solution to this problem, MAGE uses the decoupling of the process spaces by use of an automatically generated transparent proxy intercepting all calls to the native implementation shown in figure 5.10. The native component of the service is replaced by a generated proxy that exposes exactly the interface of the original component. This proxy now receives all the calls to the native methods from the JVM. Such a call is passed on to the original native implementation that is instantiated in a different process than the JVM and managed by a process server. The wrapped and sandboxed native process are referred to as the *I-Process*. Creation of the I-Processes for the Java based Grid service hosting environment is managed by a custom process manager.

The process server acts like the JVM to the native method implementations, it passes a reference to an altered JNI interface implementation to the original native code. Every reference to the JVM from the original native code is thereby

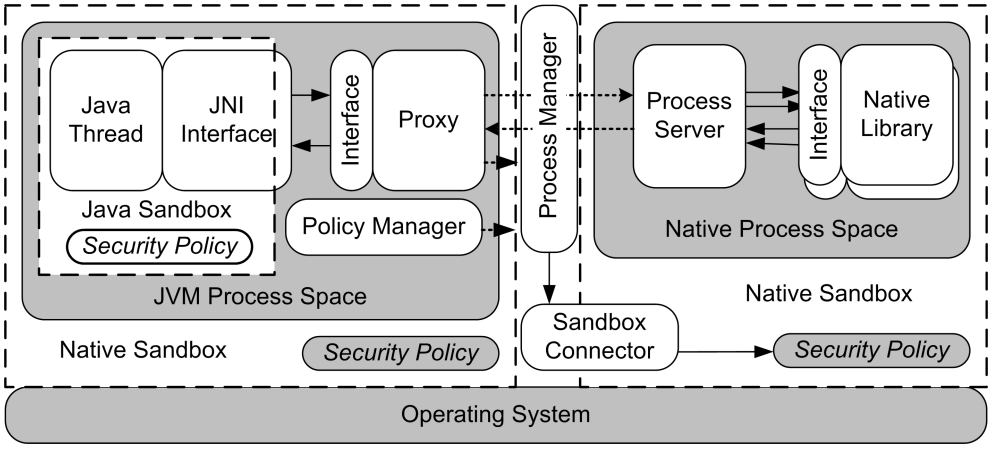


Figure 5.10: Decoupled process spaces for JNI attached native code, enabling secure isolation of native code.

intercepted by the custom JNI interface implementation. The transparent proxy and the process server communicate by means of standard IPC or RPC mechanisms, depending on the security and functionality requirements.

A second possibility to execute native code from Java is to use the Java Runtime class to create a new shell environment where the native code is then run. The Runtime class uses the ProcessBuilder to create a new shell. The ProcessBuilder itself uses a statically linked JNI native `create` method to create the operating system shell. Unfortunately, the proxy approach cannot be utilized in this case since the `create` method is integrated into the JVM. It is, however, possible to offer a custom ProcessBuilder which can sandbox a newly created shell on behalf of a service.

A number of different options for the secure execution of native code (i.e. enforcement of access restrictions to the host operating system and isolation of processes against each other) can be used in the MAGE native code isolation architecture. These options include the use of a dedicated or virtual host using Xen [23], changing the effective user ID of the child process (setuid), BSD jails [105] or system call interposition offered for example by systrace [140]. A balanced decision needs to be made between the cost (i.e. instance creation time, computational overhead, increased resource consumption) incurred on the original Java service hosting environment and the strength of security offered by the chosen method.

The above solution to inter-service security shows one possible way of protecting services from attacks within the same web service engine on which the service is running. Since with the progressive adoption of Grid technologies in the scientific and business communities, intra-engine inter-service security will become more relevant as more users will share Grid nodes. It would be best if the WSRF specifications dealt with this topic. The requirements posed at the

beginning of this section should be formulated in a platform independent way, which nonetheless binds WSRF implementations to enforce intra-engine inter-service security on all platforms. The specification should then be integrated in the WSRF specifications family.

5.2.6 Data Handling

The first implementation of Flex-SwA as a data handling component for the MAGE middleware is based on the AXIS web service framework and extends the AXIS functionality to allow web service interactions as introduced in section 4.2.6. Axis realizes a SOAP messaging framework that implements a server and a client both consisting of a set of handler chains. Each handler is capable of changing an incoming and outgoing message and passing it to the next handler in the chain. This enables pre- and postprocessing of incoming or outgoing messages both at client and at server side. Handlers pass messages in a so-called *message context* that can hold additional information. A chain is a composition of handlers and other chains. Three chains are predefined in the Axis client and server: the transport chain, the global chain and a chain where the service resides. The server-side engine has the structure shown in figure 5.11(a).

When a message arrives at the server, it is passed through these three chains which handle incoming messages. First, it is put in a message context which is forwarded to the transport chain. This chain handles transport specific issues, like the protocol being used to send the SOAP message (HTTP by default). Then, the message context is forwarded to the global chain implementing, for example, security policies. If the processing in the global chain took place without errors, the message context is passed to the service specific chain. Handlers in this chain may manipulate the message before it is passed to the actual service. A reply message from the service is passed along a response handler chain to the client.

Axis provides a `Call` object for service invocation, which handles creation of a message and invocation of a service. After starting the invocation, the client-side message processing takes place (as shown in figure 5.11(b)). The `Call` object contains the message context which again is passed through the chains. The concrete set of handlers for the various chains is determined by a XML based configuration file (`server-config.wsdd` at the server side and `client-config.wsdd` at the client side) that contains a description of the handlers and their order of invocation for the different processing chains.

The implementation of the Flex-SwA data handling component contains custom message handlers that are included in the global handler chain of the client as well as the server. The purpose of the client side handler is to handle the serialization of `Outport` or `Reference` objects that are passed to the client as invocation parameters for a web service method. A Reference object already contains information about the location and the concrete protocol to use for exchange of binary data. References are not limited to contain contact information

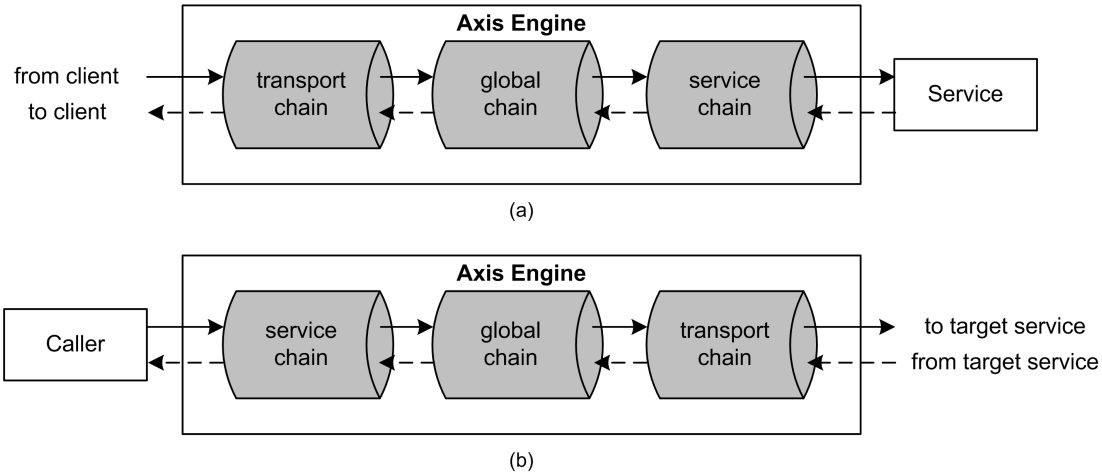


Figure 5.11: AXIS handler chains at server and client side.

for just one protocol, they may also contain several reference specifications if access to a data object is provided through several protocols (such as a direct TCP server, a P2P content distribution protocol such as the BitTorrent protocol and through a GridFTP server on the Node of the client).

If the choice for a suitable protocol is left to the middleware, the client requests the service interface description from the service provider. For this purpose, the AXIS framework implements special request handlers that emit the WSDL description of a service. Such a request is an HTTP GET request for the endpoint URL of a service with the addition of a single parameter `?wsdl`. The Flex-SwA implementation contributes a special handler to the processing chain for these service interface requests, that post-processes the service description to include information about the protocols supported by the service provider and their priority ordering. This information is added transparently as annotations in the WSDL document to keep them transparant to regular web service clients that do not support Flex-SwA. The Flex-SwA infrastructure is in addition to its application in the ad hoc Grid middleware also useful in a pure web service environment. Every AXIS deployment can be configured to use Flex-SwA. Clients and service containers of the MAGE middleware are configured to support Flex-SwA by default.

A behavioral policy is associated with each service deployed in the AXIS container by incorporating policy information in the deployment descriptor of the service that gets introduced in the `server-config.wsdd` file of the engine. This policy information is managed by a `PolicyManager` instance that handles all policy related management functionality for the platform. Policy specifications may also be changed during runtime of the engine.

When a Flex-SwA message is encountered by the Flex-SwA request handler in the global processing chain of the service container, four different courses of

action are feasible according to the current policy for the target service of the message:

1. If the service policy is eager and non-overlapping, the transmission of the attachment with the highest priority begins. Then, the attachment with the next lower priority is transmitted until every attachment has arrived. After the transmission is completed, the service is started.

2. If the service is configured to use an eager and overlapping policy, a thread is started to initiate the transmission of the attachments in priority order and then the service immediately starts.

3. If a lazy, overlapping policy in blocking mode is used, the service is started immediately. If the service needs an attachment, it blocks until the attachment is completely transferred and then proceeds execution.

4. If a lazy, overlapping policy in non-blocking mode is used, a thread is started which prepares the transmission of the attachments. Thereafter, the service begins execution. If the service needs an attachment, the thread is triggered to start the transmission concurrently.

Actual handling of the data transmission is delegated to a protocol handler that in turn is resolved according to the preferences of the client, specified in the reference set and the preferences of the service container specified in the global or service specific policy.

Figure 5.12 shows a sample policy for two services. By default, every service supports the transport protocols TCP and GridFTP. TCP is used as the preferred transport protocol if it is supported by the client, since it has the highest priority value assigned. The attribute `impl` describes the class file to use for the implementation of the protocol (package names are omitted for brevity in the example). While these default parameters have a platform scope, the following special configurations are expressed for the two services:

`LazyService` uses the transmission mode `lazy` in a non-blocking manner. The handling of data transmission and service start is `overlapping`. The service expects two attachments. The first one with the highest priority supports three protocols. The second one uses the standard protocols and has the lowest priority.

The `EagerService` uses the transmission mode `eager` and is `non-overlapping`. It expects two attachments to be transferred with the standard protocols. Since eager and non-overlapping means that all attachments have to arrive before service start, a priority does not have to be specified.

Figure 5.13 shows the client side code required to transmit a file using FlexSwA via a simple TCP protocol. The client in this case determines the transport protocol by requesting a special connector (`SocketConnector`) that uses a local TCP server to provide the file referenced in the `FileOutport` instance `fop`. The

```
<policy ...>

 <protocols>
  <protocol name="SOCK_STREAM" impl="SocketHandler"/>
  <protocol name="GRID_FTP" impl="GridFTPHandler"/>
  <protocol name="RFT" impl="RFTHandler"/>
 <protocols>

  <default>
    <protocolRef name="SOCK_STREAM" priority="30"/>
    <protocolRef name="GRID_FTP" priority="20"/>
  </default>

  <service name="LazyService">
   <operation name="lazyOperation">
     <transmission type="lazy" blocking="false"/>
     <processing type="overlapping"/>
     <attachment id="1" priority="15">
       <protocolRef name="GRID_FTP" priority="1"/>
       <protocolRef name="RFT" priority="2"/>
       <protocolRef name="SOCK_STREAM" priority="3"/>
     </attachment>
     <attachment id="2" priority="7"/>
    </operation>
  </service>

  <service name="EagerService">
    <transmission type="eager"/>
    <processing type="non-overlapping"/>
    <operation name="lazy">
     <attachment id="1"/>
     <attachment id="2"/>
    </operation>
  </service>
</policy>
```

Figure 5.12: Sample policy for an eager and a lazy service using Flex-SwA.

```
...
QName qnRef = new QName(
  "http://infrastructure.flexswa.fb12.de",
  "Reference");
...
File f = new File(localPath);
FileOutport fop = new FileOutport(f);
Reference fref = FlexSwA.getReference(
  new String[]{"de.fb12.flexswa.SocketConnector"},fop);
call.addParameter("ref",qnRef,ParameterMode.IN);
call.setReturnType(qnRef);
Reference ret = (Reference) call.invoke(new Object[]{ref});
...
```

Figure 5.13: Client code required to transmit a file via Flex-SwA, using a simple TCP protocol for binary transmission of the file.

specification of a connector list may be omitted from the reference creation function: if instead a service endpoint address or a call object with previously set service endpoint reference is passed to the method, the appropriate protocols are automatically determined by querying the service policy information first. Internally, the reference elements for the various protocols supported by the client are directly created by visiting all protocol handlers registered to the Flex-SwA layer.

Various implementations of the *IOutport* and *IInport* interface are provided to handle the attachment of large binary objects to a call or the synchronous transmission of data between a client and a service instance. The *IOutport* interface defines the method `getOutputStream()` to request an output stream that data can be written to, the *IInport* interface defines the corresponding `getInputStream()` method to obtain an input stream to read from. In addition to this very basic interface for synchronous data transmission, typed implementations are provided that implement collection semantics (e.g. a vector of elements). Developers may add elements to the Set at client side that are then serialized and transmitted over an external channel to the service instance, that can subsequently read the elements from the set. For more implementation details regarding the synchronous transmission of parameters between client and service instance, the reader is referred to [118].

5.2.7 Grid Process Execution Engine

The process execution engine implementation of the MAGE middleware is an extension of the open source ActiveBPEL [3] engine. Its integration into MAGE closely follows the design described in section 4.2.7. The implementation of the

engine is based on the AXIS framework like the Grid service core implementation of the GT4 toolkit that forms the basis for MAGE. The entire modified ActiveBPEL engine is deployed into the same web application container hosting the Grid service container of the MAGE platform under a different application context. Standard Grid services are visible under the application context `/wsrf` while the service endpoints of the BPEL processes are available under the application context `/process`.

A first step required to enable the engine to interact with MAGE services was to extend its web service invocation capabilities to also handle Grid service interaction. For this purpose, a custom SOAP handler was implemented that handles the resource key exchanged between the factory service and the engine acting as a Grid service client. As a first construct in the engine, a resource reference was introduced that can hold the endpoint reference to a WS-Resource instance, including the resource key returned by the factory service for the resource. The resource key for a WS-Resource is included in the internal data structures generated by the implementation of the `GridInvoke` activity. The ActiveBPEL engine enqueues web service invocations in a message queue for outgoing requests. If the custom SOAP handler discovers a resource key in an outbound message, it includes the information in appropriate WS-Addressing headers of the SOAP message to actually transmit the key to the partner Grid service container.

GridForEach

The ActiveBPEL implementation contains definition objects that represent each construct of a BPEL process specification in the engine. These definitions are extended by the appropriate constructs corresponding to the language extensions described in section 4.2.7. The most complex construct is the GridForEach construct, which is implemented by the class `AeActivityGridForEachDef`. Each definition class is accompanied by an implementation class for the construct. This implementation controls creation of instances of the inner scope and assignment of the input variables to this inner scope. Execution of the individual branches of the nested scope is then handled by the regular activity control mechanism of the process execution engine. The basic scheduler in the GFE implementation controls when the engine enters into a branch of the current execution flow. The construct also defines the outer scope of the nested scope. Therefore, error conditions in the nested scope are directly passed to the GFE implementation. Using this mechanism, the GFE implementation can track successful execution and mark input data sets processed and add result values to its internal result collection or directly to the merger service through invocation of the accumulator function of this merger service.

The partner link sets for a GFE are defined globally like regular partner links. The definition of a partner link set is accompanied by a collection container in the process specification. This collection holds individual instances of the part-

ner link set after they are initialized using the discovery service. The default implementation of the GFE construct uses a regular client for the P2P discovery service to perform service discovery based on the port type defined in the partner or resource link definition for every element in the partner link set. If the process specification instructs the engine to perform continuous initialization, the discovery operation for the partner link set is handled by a discovery worker thread. All discovery operations are handled by querying the local discovery service of the accompanying Grid service container. To serialize requests to this service, a link to the partner link set definition is added to a single discovery worker thread for all process instances. This also enables the aggregation of discovery operations for multiple instances of the partner links for the same process definitions since the same port types must be discovered for all instances. Assignment of discovery results is evenly distributed over multiple concurrent process instances by the discovery worker thread.

Notifications

Notification handling in the MAGE process engine is realized by use of the notification handling mechanism for Grid service clients in MAGE. For a notification subscription, the process engine registers a notification consumer with the MAGE middleware that is internally mapped to the process instance by the engine. The `receiveNotification` activity is implemented as a blocking activity that suspends execution of the process until a notification is received in the central notification handler. This central notification handler identifies the suspended process instance and instructs the engine to resume execution at the notification reception activity.

Process Management

The deployment of a process into the ActiveBPEL engine requires a number of files to be present including the process specification, the deployment descriptor and a WSDL description of the service and possibly external partner services. Similar to the Grid archive file containing a regular Grid service, ActiveBPEL supports deployment of the files in a business process repository (BPR), a specially structured Java archive file. The process engine watches a local folder for the addition of a BPR file through an instance of the class `AeDirectoryScanner` and then adds the business process contained in a BPR file found in the deployment directory to the deployed services. This facility is used by the process deployment service that exposes deploy, undeploy and redeploy operations for process specifications. The deployment operations have been implemented using Flex-SwA to allow actual transmission of BPR files of arbitrary size and place them in the engine deployment folder. Additional management and enumeration features of the ActiveBPEL engine are exposed through a web application inter-

face. Undeployment of a service is handled by issuing the undeploy command to the engine through its web based interface.

Similarly, the process engine exposes a list of deployed processes and their service endpoints through a web based interface. A special singleton Grid service is instantiated in the MAGE Grid service container that watches this process list of the locally deployed engine. Service endpoint information for the deployed business processes is published to the P2P information space similar to the information for regular Grid services deployed in the Grid service container. These mechanisms require active polling of a local folder respectively the list of deployed services that can cause a slight delay in the actual execution of the deployment action and the publishing of information. However, this delay of a few seconds never proved to be problematic in most application scenarios. The delay in the deployment operation may be removed from the application by using the direct deployment facility of the management interface. A delay in publishing the service endpoint information may be removed by altering the engine such that it calls back to the accompanying Grid service container signaling the change to the service registry of the Grid service container.

Security

In the current implementation of the MAGE process execution component, the process engine directly handles web service interactions. The access control and message security mechanisms of GT4 and MAGE are implemented as AXIS message handlers. They can also be configured to be used in the message handling chain of the process engine. However, some precautions needed to be taken to ensure that these security handlers act on the same configuration base in order to allow for seamless management of both application contexts.

Since the process engine interprets the process description, no native code or custom application code is executed in the process space of the engine, which would require additional isolation. Process instances of different users are by default isolated against each other by the engine, no process instance can directly access process variables of other process instances.

Robustness

The internal scheduling mechanism used for the GFE construct achieves a high level of robustness. An attempt to repeat execution of the nested sequence with a newly discovered resource set is undertaken until all input data is processed or a timeout condition is met. Depending on the complexity of the activities in the nested scope, this leads to coarse grained retry of activities. A detailed discussion of possible solutions to achieve more robust business process execution through re-invocation of activities can be found in [78]. The integration of a similar strategy into the process execution engine is a topic of future work.

5.3 Development and Management Components

This section offers a description of a concrete implementation of the Grid Development Tools for the Eclipse Platform. The current implementation of the GDT includes target system mapping implementations for the Globus Toolkit 4 and the Marburg Ad-hoc Grid Environment (MAGE). The Eclipse platform is described as "an extensible development platform and application framework for building software". It is built as an open source effort using the Java language. The Eclipse community has gained a large group of supporters actively developing core functionality as well as components. As one of the most popular Java IDE's it presented itself as a logical choice for an integrated development environment to be extended by a set of Grid Development Tools.

The entire Eclipse platform is implemented as a small core runtime environment for plugins and a large collection of plugins that provide the functionality of the platform. Plugins define so called *extension points* and implement extension points defined by other plugins to contribute their functionality to each other. The Eclipse platform supports a *headless mode*, i.e. running a so called Eclipse application without the graphical workbench user interface. A standard headless application of the Eclipse platform provides Ant functionality on the command line. In this mode, the Eclipse platform acts like a standard Ant distribution taking a build file as input and making all build tasks defined by plugins in the Eclipse installation available to the build file.

The GDT's functionality is provided to the Eclipse platform through a number of different plugins. These plugins implement the different extension points provided by the platform or external plugins. Figure 5.14 shows the plugin hierarchy implemented in the GDT. Core functional components are strictly separated from user interface components to ease the re-use of the core functionality in headless mode of the Eclipse platform.

5.3.1 Service Creation Support

As a first step towards an implementation of the GDT, the meta-model for the upper layer Grid PSM was devised. The resulting meta-model is shown in figure 5.15. The basic components of this meta-model are a *Grid project* that can contain many *Grid services* that in turn have a number of *Grid attributes* and *Grid methods*. Following the WSRF definition of a WS-Resource as the union of a stateless web service and a corresponding resource property document capturing the state data of the resource, the Grid attributes are the elements of a service forming the resource property document. Every attribute, method parameter, return type or exception thrown by a method has an associated type that can either be a simple type or a complex type. The Eclipse Modeling Frame-

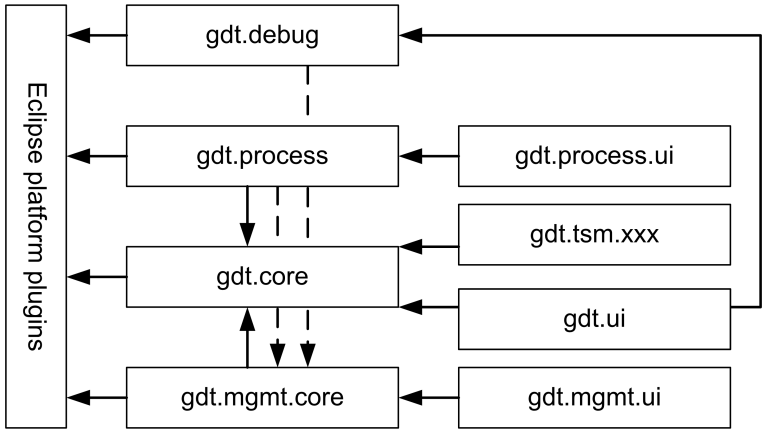


Figure 5.14: Hierarchy graph showing the internal dependencies of the GDT plugins.

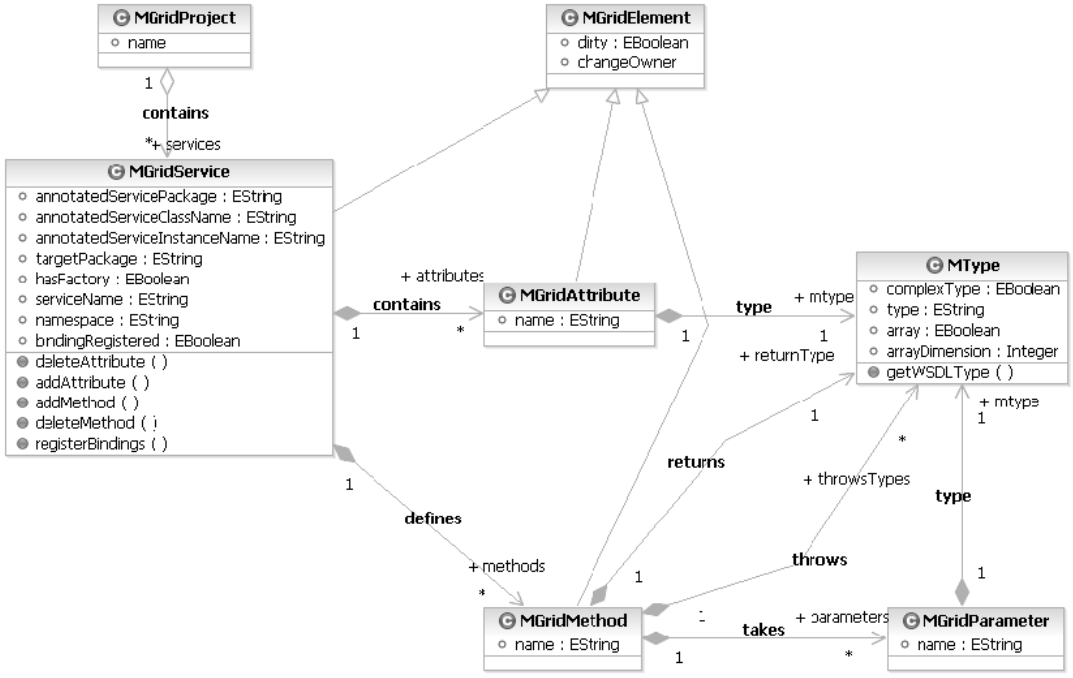


Figure 5.15: Meta-model for the upper layer Grid PSM.

work (EMF) is used to automatically generate a Java implementation of the meta-model. Instances of this representation are used throughout the GDT for representing concrete upper layer PSMs.

Several Java annotations are defined in the GDT implementation to represent the connection between application logic carrying code and the upper layer PSM of a Grid service. These annotations are `GridService`, `GridMethod` and

`GridAttribute` and can be used to annotate Java class or method definitions and field declarations. The Java annotations can be used to define additional meta-data such as the target namespace for the Grid service. The annotations are intended to be used by developers in their applications in order to mark certain logical elements for inclusion in a Grid service. In terms of the GDT, a Java class carrying the `GridService` annotation is called an *annotated service class* and may be interpreted as a model source for the upper layer PSM.

The Eclipse platform follows a workspace concept for organizing user resources. A user's workspace contains several projects that in turn contain a number of resources. Each project has a so called nature assigned that is used to filter the user interface and the collection of plugins that act on the project and the resources contained in that project. As a first extension to the Eclipse platform, a project nature for *Grid projects* (the `GDTNature`) was defined and implemented that can be assigned to any Java project in the user's workspace to allow associating the GDT plugin with the project. Apart from the project nature, the Eclipse platform assigns an ordered set of project builders to a project. These builders are invoked whenever the change of one or many resources forces an incremental or full build of the project or when the project should be "cleaned". When a new service is added to a Java project, the GDT assigns the GDT nature to the project and a GDT builder which is running after the Java builder of the Java Development Tools (JDT) plugin.

The GDT builder is the central component handling the internal transformation between the upper and lower layer PSM of a project and the central contribution of the GDT core plugin to the Eclipse platform. The builder receives a so called delta set from the Eclipse platform. This delta set contains a reference to all resources that were changed since the last build was performed on the project. For an incremental project build, this is the set of resources actually touched by the user (or an action by the user inducing the changes on the project). The GDT builder now performs two operations in sequence. First, a delta visitor is used to check every resource in the delta set for the project whether it is a model source. Second, the model transformations and emitters are invoked to push changes in the model into the actual resources. Every element in the Grid service model inherits its interface from the GridElement interface. Part of this interface is the ability to query the element for its state. Modification operations on a model element set a modification flag, the element is assumed to have a *dirty* state. After performing all model transformations and code generation, the changes are assumed to be reflected in the target model or source code and the dirty-flag is cleared for all model elements. This mechanism allows a fine-grained decision on whether to run a particular transformation or code emitter based on fine grained modification information in the Grid service model.

To perform these operations, the GDT builder needs to make a connection between resources and the elements of the project model. Figure 5.16 shows the internal binding model used for this purpose. The binding model defines a binding

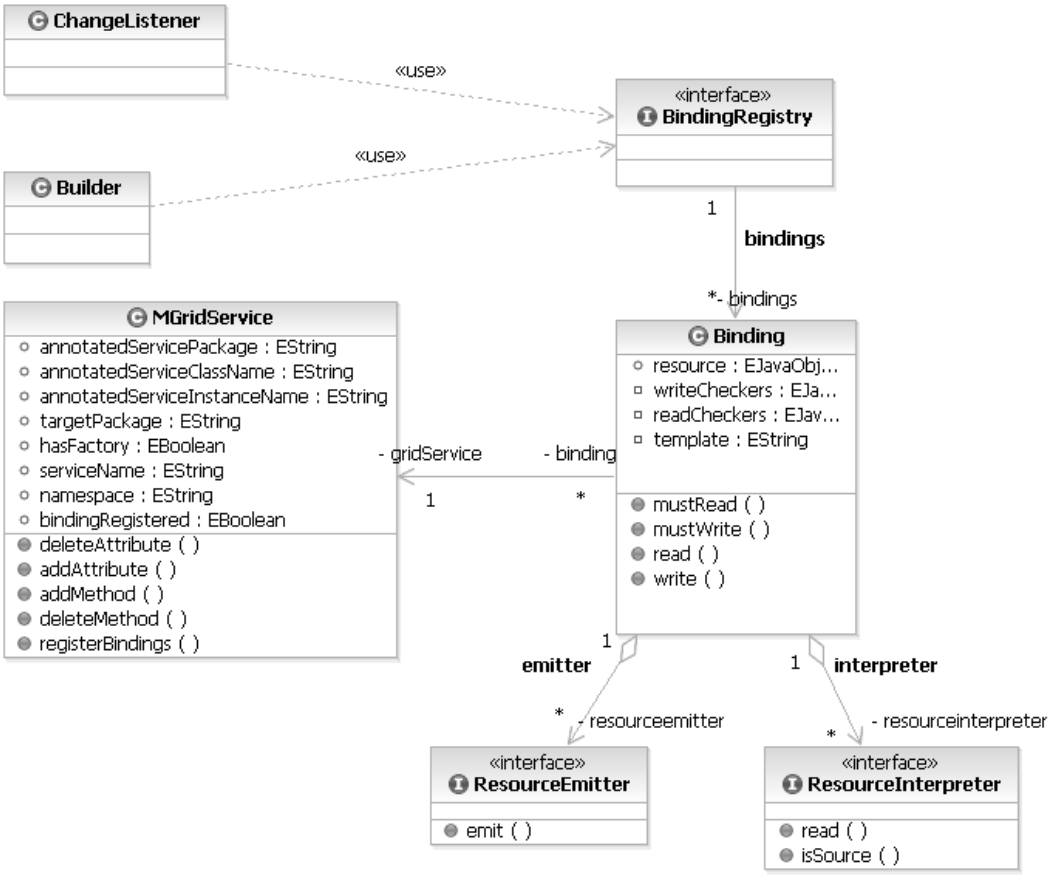


Figure 5.16: Internal binding model between service, resources, emitters and interpreters.

element that is directly connected to a workspace resource and holds references to a service model element, a resource emitter and a resource interpreter. Binding elements are collected by the GDT plugin in a binding registry that can be used to look up bindings for a given service model or a given resource. To decide whether a resource is a source of model information, all resource interpreters defined for the GDT must implement a method `isSource` that is invoked on resources that have no binding information attached to them. Based on the result of this heuristic part, a binding between the resource and the particular resource interpreter is constructed and registered for the project. Apart from this heuristic part, all resource interpreters implement a transformation part handling the transfer of information from a resource into the corresponding model representation.

The GDT uses the annotation processing component of the standard Java Development Tools (JDT) to process annotations in Java classes. For this purpose, the GDT registers an annotation processor with the JDT. The JDT Java builder is invoked on the resources of a project prior to the GDT builder, and the an-

notation processing component of the Java builder hands any annotations to the GDT annotation processor. This processor registers the annotation information for every resource in a Grid annotation registry. Information in the annotation registry is managed on a per project basis. This registry is then queried by both the heuristic as well as the transformation part of a resource interpreter to decide whether the resource is an annotated service class or to actually fill the service model from the definition of a class. The resource interpreter makes direct use of the structural information of the Java class under inspection. The JDT offers the ability to easily access structural information reflecting the declaration of a Java class through the document object model for Java elements. Other resource interpreters use internal models for other kinds of resources such as XML schema files and WSDL files provided to the Eclipse platform by other plugins such as the Web Tools Project [56] or the EMF [55].

Most code emitters of the GDT are implemented using Java Emitter Templates (JET) technology, a part of the Eclipse Modeling Framework. JET uses a syntax similar to Java Server Pages, allowing to use regular Java code within special code sections of the template. Everything outside of the special code tags is literally copied into the output. Since there is a trivial mapping between the lower level PSM and the actual source code for the application, the GDT can directly emit source code from the upper layer PSM. This is also true for non-Java resources such as deployment descriptors and WSDL files for the target service. JET is supplemented by *JMerge*, an EMF component allowing to merge newly generated versions of a Java resource with previous versions that might contain user modifications. JMerge transfers the user modification into the newly generated resource preventing loss of user source code. Apart from the template based code emitters, the GDT implementation relies on existing code generators for the GT4 framework.

The generators are implemented as a combination of Ant build scripts and small code generators implemented as Java programs that get called from the Ant build files. These tools are used to generate, for example, stubs for the resulting Grid service. They perform their work in a rather long running process (around 20 seconds) that is hard to include in an automatic and incremental project build. Fortunately, those long running generation steps target only automatically generated platform binding code a human developer is unlikely to ever modify. Developers will rather focus on changing the application logic or the code corresponding directly to the lower layer PSM generated by the template based approach. Since the stub generation process is such a long running operation, developers are given the opportunity to manually start the stub generation process for a service as a last development step prior to service packaging and deployment or when they see substantial changes in the service model that requires (re-)generation of the stubs. The resulting source code of the stubs is automatically added to the project and can be modified by the developer within the workbench. The second long running operation that can directly be triggered

```
<target name="import.math">
  <gdt.generator
    tsm="gt4"
    project="TestService"
    globus="${env.GLOBUS_LOCATION}"
    annotated="path/to/Math.java"
    service="Math">
    <preimport dir="Math_extra">
      <include name="src/**/*.java"/>
    </preimport>
  </gdt.generator>
</target>
```

Figure 5.17: Sample Ant target using the GDT Ant task.

by the user is the final packaging of the Grid service for deployment. The packaging operation for both target systems implemented also uses existing Ant scripts and integrates them into the workbench.

All transformation components of the GDT are split between the GDT core plugin and so called *target system mapping* modules (TSM). The core plugin provides the meta-model for representation of Grid services as well as common and generic services such as annotation processing and model registries, while the TSMs contribute their interpreters and emitters to the core plugin.

Developers are supported in the creation of new services by a multi-page wizard. The GDT user interface component queries the platform for all available TSM contributions. The first page of the wizards queries common information about the new service such as its target namespace, name, the package and class name of the annotated service class to be created. The GDT user interface queries the Eclipse platform for all available TSM contributions and collects a set of TSM identifiers that the user may select a concrete target system from. This selection determines additional wizard pages that are contributed by the individual TSMs and may allow the user to specify additional choices and values. After finishing the wizard execution, the service model is created and all necessary project settings are changed according to the user's choices. Most importantly, the TSM to be used for transformation is associated to the service in the project. This TSM is then used to generate all service artifacts including a base implementation of the annotated service class that the user may fill with his/her application logic.

The addition of a service to a project does not rely on the graphical user interface components. The GDT contributes a custom Ant task `gdt.generator` and a headless application `de.fb12.gdt.Generator` to the Eclipse platform that can be used without the graphical workbench user interface. Their basic intention is to enable the user to automatically create a Java project for an annotated class or add the service to an existing project, generate the target system specific artifacts

```
java -jar startup.jar
    -application de.fb12.gdt.Generator
    -data <workspace location>
    -project TestService
    -tsm gt4
    -annotated path/to/Math.java
    -globus $GLOBUS_LOCATION
    -import Math_extra/src/**/*.java
```

Figure 5.18: Command line invocation of the GDT.

and package the service. They also offer the ability to initialize a new workspace, and to automatically import any number of Java libraries and Java source files into a service project. This functionality is intended for use in automated build or test environments that are very common for large Grid applications. Both components can be regarded as different user interfaces to the core components of the GDT. Therefore, the implementation of both components is part of the GDT UI plugin. It relies on functionality for project creation and modification that is provided by the core plugin. Figures 5.17 and 5.18 show an Ant target and the command sequence used to invoke the GDT from an Ant build file or the command line, respectively. In both cases, the annotated class `Math.java` is added to the project `TestService` and the service is built and packaged as a service for the Globus Toolkit 4. Additional sources from `Math_extra` are imported into the project prior to service generation.

On a Pentium 4 Mobile 1.7GHz system with 1GB of RAM running Windows XP, the transformation from upper layer PSM to lower layer PSM and the creation of the corresponding artifacts takes between 1 and 2 seconds. This time constraint is acceptable for performing interactive development work on the annotated service class or the generated lower layer PSM without too much interference to the developer. The existing build tasks for stub generation and packaging of the service are long running jobs that take between 15 and 25 seconds to finish, which is another reason to only carry them out on explicit request by the user. Headless operation and the use of the Ant target from the command line adds approximately 10 seconds for the initialization of the Eclipse platform components. The overall time required to automatically import an annotated service class into a newly created workspace and project, generate, build and package the service is around 50 seconds. In the automated test environment for the GDT, the same task takes about 35 seconds on a desktop Pentium IV 3GHz systems with fast S-ATA drives. In both cases, the performance of the generator supports the intended use cases for automatic service generation and interactive development.

5.3.2 Process Creation

A distributed, graphical process editor for Grid processes based on the BPEL language was implemented for the MAGE platform. The implementation of this process editor strictly uses the MVC pattern and is based on the Eclipse Graphical Editing Framework (GEF) for the presentation layer. The GEF, like many other GUI toolkits, is built using the MVC pattern, making it easy to implement the GDT process editor supporting the overall requirements and design decisions presented in section 4.3.3. The implementation of the model sharing component is based on the Eclipse Communication Framework (ECF) [54].

Core Model

There are three basic types of elements in the core process model. The base class of every model element is the class `Element`. Two direct descendants of this base class are `ContainerElement` and `Connection`. Containers may directly contain other model elements. An example for such a container is the `Sequence` class that represents a BPEL sequence of activities. Sub-classes of the connection class are used to represent links between activities in the process such as the links between activities in a BPEL Flow (a flow is not ordered like a sequence but a collection of activities that are executed based on transition rules). A globally unique ID [112] is assigned to every element in the process model, allowing to uniquely identify the elements even across different nodes sharing a single process model through the model sharing component. Conceptually, every element of the process model represents a collection of attributes. The `Element` class implements the `IPropertySource` interface allowing the Eclipse platform to directly display the attribute values of a selected model element in the standard property view. The root element of every process model based on this meta-model implementation is an instance of the `Process` class. The GDT core model implements elements to represent a process following the BPEL 1.1 specification.

The `IPropertySource` interface defines operations to retrieve a list of `IPropertyDescriptors` and access methods to set the value of a particular property or to get a property value from the model element. Every element sub-class in the model collects the property descriptors from its super class before adding own property descriptors to the list of return values. The setter and getter methods for property values handle the local properties for the concrete Element class and delegate to the methods of the super class for unknown properties. Properties are identified by String values passed as property values. As a means of selectively displaying certain element properties, the core process model allows to select a set of filter rules on the model instance. Before returning the result list in the implementation of the `getPropertyDescriptors` operation, the list is filtered by a `PropertyVisibilityFilter` that removes every property descriptor defined in the exclusion list defined for the element. While the properties are

filtered from the view, the property access methods of the element still provide access to filtered attributes. This allows wizards automating tasks for the user full access to every element of the process model even if it is not directly displayed for the user.

Plug-ins may contribute to existing property visibility filters by specifying a `VisbilityFilterContributor` in their implementation of the `modelExtension` extension point of the core GDT process editor plug-in. Filter contributions are identified by a filter ID and may define a list of property IDs per element class. The contributed filter lists are aggregated by the `PropertyVisibilityFilter` and applied to the element based on the element class and the selected filter ID. Visibility filters may only remove elements from a view by adding exclusion rules to a filter set. They must define a new filter set under a new ID if they want to expose properties filtered by another filter contribution. This ensures that the original filter contribution is not altered in a way that properties are exposed to the user that were intended to be under control of a wizard component.

The model needs to be serialized and deserialized for storage and transmission. Serialization of the model elements is handled by proxy classes that implement the serialization capability for the core model elements. The proxy classes handle storage of the model elements. Therefore, they are referred to as *storage proxies*. Every storage proxy instance holds a reference to an associated model object. A storage proxy instance for a concrete instance of a model element can be obtained from a factory that receives the model element object in the proxy creation request and constructs or returns the associated proxy instance. Storage proxies for child objects of a container model element are constructed recursively. The default implementation of the storage proxy elements implements the methods `toDOMElement` and `fromDOMElement` in the `IStorageProxy` interface. These methods return or interpret a tree of XML elements representing the elements and connections in the process model. The structure of the resulting XML elements and attributes is governed by the concrete implementation of the `IStorageProxy` instances returned by the proxy factory.

The XML element names in the serialized form of the core model are fully qualified. The namespace of an element is used by the standard deserializer to determine the Java package of the class corresponding to an XML element. Namespaces for contributed model extensions can either be specified in the extension definition of the model contribution or a standard package to namespace mapping is used by the serializer and deserializer of the elements.

The storage proxy interface supports serialization of the internal model to various destinations. In addition to the methods allowing to query the model for a single DOM element, methods are defined to retrieve an array of DOM documents together with information about the placement of the individual documents within a path hierarchy and direct retrieval of the serialized form of the documents as Java streams. A last option of serialization is presented through a set of `serializeTo` methods that take a target output location (e.g. a folder in

the local file system) as input parameter and serialize the entire model to that location. This extended serialization options are typically only implemented in the storage proxy implementations of the Process element, for other elements an `OperationNotSupportedException` is thrown.

Similar to the regular serialization implementation for simple storage of the process model, a concrete target system mapping implementation may perform serialization of the model into the required process representation by providing another proxy factory. The basic serialization implementation queries all elements for their unfiltered properties and serializes them into attributes and child elements, enabling storage and retrieval of the complete model information.

Presentation Layer

The GDT process editor contributes an implementation of a graphical editor as its presentation layer implementation. Its purpose is to provide a graphical representation of the process model to the user that visualizes the process structure and lets the user add, remove and rearrange components, connect them and edit the properties associated with the process activities and other elements. GEF is separated into two parts: the Draw2D API for efficient painting and layout of *figures* and the GEF API adding editing capabilities on top. A figure is the visual representation of a corresponding model element. So called `EditPart`s connect figures and model elements.

EditParts for model elements are created by an `EditPartFactory` associated with the editor component. Edit Parts in turn create figure elements for display of the associated model elements in a view. A factory based approach for creation of `EditParts` and `Figures` for model elements is used throughout the entire GDT process editor. This allows to flexibly replace figure implementations used by the editor while keeping the EditPart implementations. Implementations of the EditPartFactory as well as the FigureFactory are extensible through standard exension points. Based on the type of the associated model element, the factories try to resolve a factory contribution capable of creating a corresponding `EditPart` or `Figure`. The currently set detail level for the process editor is also passed on to the factory contributions enabling them to contribute elements for the right level of detail.

Editing is the most complex operation performed by the EditParts. The Eclipse platform abstracts the source of an operation as a Request, actual modification of the model elements is then performed by a `Command` retrieved from the target edit part. During an editing operation, the edit part also shows feedback (such as drag handles for move and resize operations) to the user. EditParts do not handle editing directly, rather they delegate editing to EditPolicies associated with the part. Before returning a command or signaling approval to an edit request, the core model is queried for any edit locks set on the model for example by the model sharing component.

The EditParts are property change listeners to their associated model elements. The view corresponding to an edit part is updated based on the property change event fired by the model element after it was modified by a command. Other components and views on the model elements are also capable to update their display based on the property change events emitted from the model element.

An Eclipse workbench window shows a so called *perspective* that is a collection of multiple views on the workspace of the user or currently selected elements. The MAGE GDT process editor uses the properties view in addition to the graphical representation of the process model. The properties of a particular model element are accessible to the user through a tree that allows editing of the individual properties. The set of available properties is determined by the previously described view filters.

Model Sharing

The model sharing implementation of the MAGE GDT process editor uses facilities provided by the Eclipse Communication Framework (ECF) [54]. In order to collaborate, the users of the process editor join a collaboration channel (access protection to the channel may be set up by the collaboration initiator). The node of the collaboration initiator also acts as a coordinator to the collaboration. After joining the collaboration, a new editing partner may request the model from the channel. The initiating partner then serializes the model and transmits it to the newly joining party. For the new editor the model is deserialized and used as input to the graphical editor.

The underlying communication channel implements a protocol that ensures reliable message transmission to a selected partner or to all communication partners in the channel. Now, two editors are connected through this communication channel, their editors show a view on the same (shared) process model. Actual update of the distributed model happens by relay of the edit commands upon their execution.

For this purpose, every command is derived from a `BaseCommand` class that triggers serialization of the command and transmission to other connected editors in its `execute` method, if the editor is connected to a channel. Every command implementation is required to call its super classes `execute` method to ensure transmission of the command. Every command implements the `IAdaptable` interface and returns a serializable and transmissible version of the command to the model sharing component that transmits the command to other editors in the channel. References to model elements are encoded using the globally unique identifier of the element.

Upon reception of a command, the model sharing componente schedules the command for execution as an asynchronous task in the user interface thread that is the owner of all edit parts and views. Execution of the transmissible command

causes property changes just like locally requested commands do, leading to a property change notification to the associated edit part and the update of the view.

Extension Mechanism

Plug-ins can extend the core process editor by contributing to the extension point `de.fb12.gdt.process.modelExtension`. Every contributed extension must use a unique identifier and specify a display name to be used by the various general user interface elements in the core editor implementation. For example, this capability is used during the creating of a new process model. In this case, all available `modelExtensions` are collected using the standard Eclipse mechanisms for extension handling. The list of possible extensions is then presented to the user.

Every model extension represents a collection of various contributions under a common extension ID. Implementations can provide factories for model elements, edit parts, figures for the view and tool definitions that represent elements in the tool palette of the graphical editor.

Factories are used throughout the implementation of the process editor to handle instantiation of the various classes of model elements, visual representations and other types of internal elements. A typical use case for these factories is the creation of an instance of a class, representing a certain object in another aspect of the editor implementation such as the creation of an edit part for every model element that is to be displayed and manipulated in the graphical view. In that case an edit part factory is asked to return an edit part encapsulating editing capabilities for a concrete model element. All factories in the editor core are extensible. If the object creation request can not directly be answered by the core factory, a contributed factory is resolved based on the extension ID passed in the object creation request. Then, the object creation method delegates to this factory that can fulfill the request.

MAGE Target System Mapping

A target system mapping for the MAGE process execution environment has been implemented. The MAGE process execution engine implementation (see section 5.2.7) is based on the ActiveBPEL engine. The actual transformation into the necessary process description files, deployment descriptors and WSDL interface descriptions is implemented as a set of storage proxies with a corresponding storage proxy factory. Serialization of the artifacts is accessible through the `serializeTo` methods of the Process element storage proxy of the MAGE target system mapping component. The standard serialization mechanism of the core process editor is fully capable to generically serialize and deserialize a process model containing elements from any extensions including the extensions for

the MAGE platform. The target system mapping is just required to implement the serialization to the MAGE process execution engine. All contributions of the MAGE target system mapping are collected under a single extension ID (`de.fb12.gdt.process.mage`).

As a model extension this target system mapping contributes additional elements such as the GridForEach construct (see section 4.2.7) to the core model. The model extensions are accompanied in the presentation layer by additional edit parts as well as figures for graphical display. During the implementation of the MAGE extensions, no additional code was required to enable the inclusion of the extensions in distributed process editing. All editing functionality could be implemented using standard commands available in the core process editor that are by default usable on the distributed and synchronized model.

5.3.3 Interactive Debugging

The Java Development Tools for the Eclipse platform provide an interactive debugger for Java classes. The Java Platform Debug Architecture (JPDA) [162] defines means for communication between a debugger and a Java virtual machine (JVM). These definitions already allow the Eclipse Java debugger to attach to a JVM running on a remote computer - the so called *target JVM*. As a prerequisite, the target JVM must be instructed to load a special *agent library* that handles communication with the debugger and interfaces with the local JVM to control its execution. This agent library implements the Java Debug Wire Protocol (JDWP) for communication with the debugger. As a means of interfacing with the JVM, the JVM Tools Interface (JVM-TI) is implemented by the agent library. The tools interface allows any agent library to register functions to be executed prior and after certain operations of the JVM take place, such as for example class instantiation or method invocation. The JVM-TI also allows for inspection of attributes and values managed by the JVM. By these means, the Java debugger of the Eclipse platform can control the execution of any Java process running in the JVM and inspect all values in the application.

For the ad hoc Grid, this means running the Grid service container in a JVM with loaded debug agent library. Even though the JDWP provides basic means to protect the connection between debugger and target JVM, running the JVM in debug mode allowing remote connections should be avoided in a production environment. Rather, a dedicated test instance of the Grid service container should be used for debug purposes. This also avoids accidental interference with a running production application in the Grid service container. The ad hoc Grid service container is considered to be a remote container even if it runs on the same physical node but in a different JVM.

The primary use case for the interactive debug support component of the GDT is to debug a service currently under development. Assuming that the packaged service has been deployed into the remote Grid service container, the

source code to the service classes is available to the Java debugger in the local project in the user's workspace. But debugging a Grid service or Grid application often also requires to step into platform components. To support the user in this operation, the source code of these infrastructure components must be attached to the process under inspection by the debugger. For this purpose, the GDT contributes an implementation of the interface `IPersistabelSourceLocator` to the platform by implementing the platform extension point `org.eclipse.debug.core.sourceLocators`. The platform passes an instance of `IStackFrame` to the source locator's `getSourceElement` method to retrieve the source element associated with the current point of execution. The stack frame information handed to the source locator contains information about the suspended thread that is handed to the GDT debug information service to enable the information service to identify the correct thread for information retrieval.

The source elements can be directly retrieved from the user's workspace for all Grid service classes under development. For platform components, the GDT source locator first queries the remote Grid service container for the actual version of a module being executed. If the source code for a Java element is not available locally, the GDT debugging service queries other nodes for the right source code library. Source code is again cached locally in order to allow for faster retrieval in the future.

The source code repository is implemented using the P2P discovery service of the MAGE platform. Modules of the MAGE platform are packaged as Java archive (JAR) files. Those archives contain a meta information folder that holds an archive manifest file. During the build process for the platform, a binary archive and the corresponding source archive containing the source files for every class in the binary archive are created. A globally unique UUID [112] is generated for every module and included as unique identifier in the manifest of both the binary as well as the source archive. Every MAGE Grid service container can optionally contain a source archive for every binary component that is being used for a concrete implementation. The inclusion of source archives is advised for Grid service containers intended for debugging since this guarantees availability of all source archives for the currently used platform components. The container automatically registers all source archives in the underlying P2P network using the UUID of the component as a primary key. Similarly, the GDT environments that retrieved a source bundle register their copy of the bundle in the P2P network for retrieval by other development environments.

Implementation of the GDT information service as a regular Grid service running in the same container under control of the Java debugger is probably not feasible. There are two situations the GDT must handle:

- The Grid service container uses a *single version* of every component, the remote debugger does not need to distinguish the class version for a concrete

stack frame.

- The Grid service container contains *different versions* of at least one component, so the remote debugger might need to distinguish between class versions for a concrete stack frame.

In the first case, version information for all components in the remote container can be prefetched before the debugging session is started. Version information is then cached in the local GDT environment per connection to a container and the right source versions are handed to the Java debugger based on the current stack frame and the pre-allocated version information. In this approach, it is also possible to prefetch source archives for all components used in the remote Grid service container. With the platform's ability of hot deployment of services and service components, this approach might, however, be dangerous. Version information for services and components deployed after retrieval of the version information must still be gathered during runtime. Therefore, a communication mechanism for information retrieval is needed that is independent of the current state of execution control of the target JVM.

In the second case above and to handle the previously described situation of component additions to the Grid service container, retrieval of information during runtime of the debugger is required. However, retrieval of detailed version information about a component for the actual current stack frame may be required while a thread of the target JVM is in suspended state. The suspended thread of the target JVM may, for example, handle all incoming connections. In this situation, the Grid service container is unable to handle the debugger's request and the debugging environment runs into a deadlock situation. In this case, the information channel for the debugger needs to be independent from the execution of Java threads in the target JVM.

The GDT debug information service is therefore implemented as a JVM-TI agent library that is loaded in addition to the regular debug information library. A simple binary wire protocol is used for exchange of information between the debug information service and the client in the debugging framework since the service cannot make any use of the infrastructure components for connection handling and Grid service protocols. This approach seams feasible since debugging of a platform is a use case with limited requirements on interoperability and configuration of the environment compared to regular operation of a Grid environment.

To actually associate the archive version information to a Java class referenced in a stack frame visible to the debugger, the GDT debug service requires additional information that can not directly be queried from the JVM. The class holds a reference to its class loader but not to the underlying archive the class was loaded from. Also, the standard `ClassLoader` interface does not allow for retrieval of this information. All hot deployable components of the MAGE platform are fortunately loaded by custom class loaders as described in section 5.2.4.

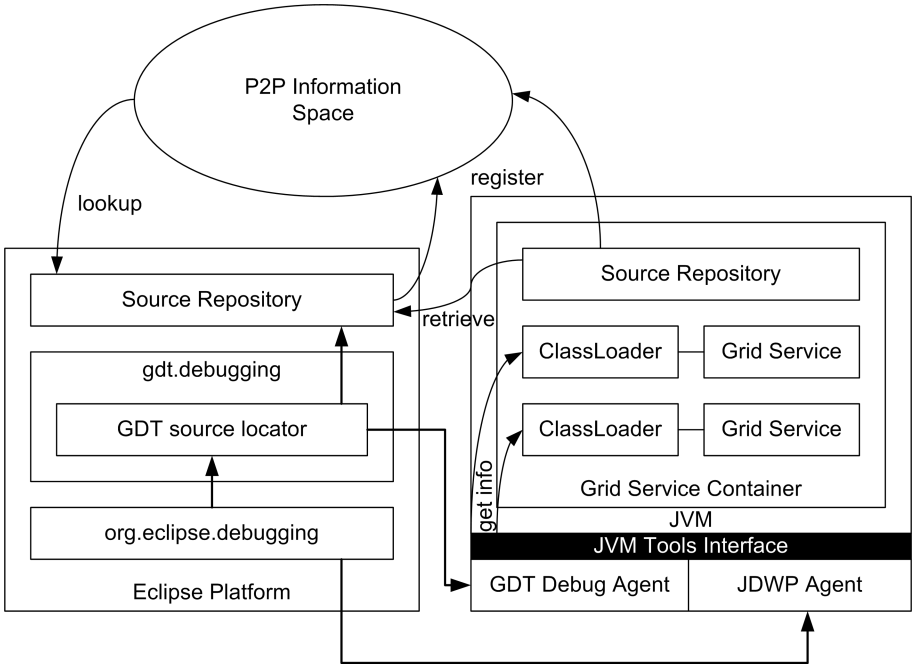


Figure 5.19: GDT debugging infrastructure components and their connections.

Those class loaders implement the interface `IDebugVersionInformationSource` that provides the method `getClassVersions`. The GDT debug information service calls this method after instantiation of a class loader to receive a map assigning an archive version ID to every class available from the class loader. This information can then be retrieved from the GDT debug information service when the debugger needs the class version information for a stack frame.

The components and connections of the resulting GDT debug support infrastructure are shown in figure 5.19. In its current implementation, the GDT debug infrastructure exposes module information through the GDT Debug Agent loaded in the JVM under debug control. Communication between the GDT source locator and the GDT debug agent is implemented using a high level connection oriented binary protocol. It is possible to restrict access to the GDT debug agent based on the network address of the client and using simple password authentication in the protocol. In a scenario where users request operation of the debug infrastructure fully integrated with the Grid middleware and the service access and management components, the GDT debugging information Grid service may be deployed in a second Grid container running on the same node the inspected Grid service container is deployed on. The standard Grid management procedures and access control mechanisms, including Virtual Organization management and role based access privilege delegation - may then be applied to this second container. In this scenario, communication between the custom GDT debug agent and the GDT debug information Grid service is based on the same

protocol that can also be used directly between GDT source locator and GDT debug agent. A WS-Resource instance of the GDT debug information service represents a connection between the GDT source locator and GDT debug agent.

5.3.4 Grid Management Components

The MAGE management components have been integrated into the Eclipse platform. In addition to running the well known collection of plug-ins in the standard Eclipse workbench, the framework supports the definition of a sub set of plug-ins that make up a custom application, called an Eclipse *Rich Client Platform* (RCP) application. The RCP mechanism offers the ability to use the user interface components for the MAGE management modules both in the integrated development environment and in a stand-alone application that only provides a Grid management user interface without development support.

As a basic service, the entire MAGE implementation has been integrated as an Eclipse plug-in. This plug-in enables the Eclipse workbench to internally start a MAGE container instance that can act as a notification consumer. This ability is for example used in the integrated P2P discovery service.

For the integration of the discovery module, two views have been implemented. One realizes an input dialog for user specification of an initial Grid node to act on, as well as information about the group the user wants to join for management. After selecting a group, this input dialog view may be used to specify a discovery query that is internally passed to the P2P discovery service on the Grid entry node. The regular P2P discovery client is used for this action and discovered results are collected via notification. The second view displays a graphical representation of the Grid nodes that have been returned to the discovery client. Relationships between the nodes are constructed based on connection information returned by the discovery service. For the FreePastry/Scribe based implementation of the discovery service, the connections represent the overlay multicast tree constructed by the Scribe layer of the P2P framework. The Prefuse [138] graph visualization framework is used to layout and render the graph. Additional information about a Grid node is presented to the user as a tool tip windows when the mouse cursor is placed over the corresponding node in the graphical view.

Access to the hot deployment service of the platform is another example for the integration of the management components into the Eclipse workbench. A so called *action set* is contributed to the workbench that combines Eclipse representations of the (re-)deploy and undeploy operations of the hot deployment service. Individual actions in the action set are contributed as menu items in a global *Grid* menu in the menu bar. They are also contributed to context menus based on the resource type the context menu is displayed for. If the user opens the context menu for a Grid service archive in the resource tree view of the workbench, he may select, for example, the *deploy* action in the Grid submenu. This action either displays an additional dialog that lets a user add the endpoint address for

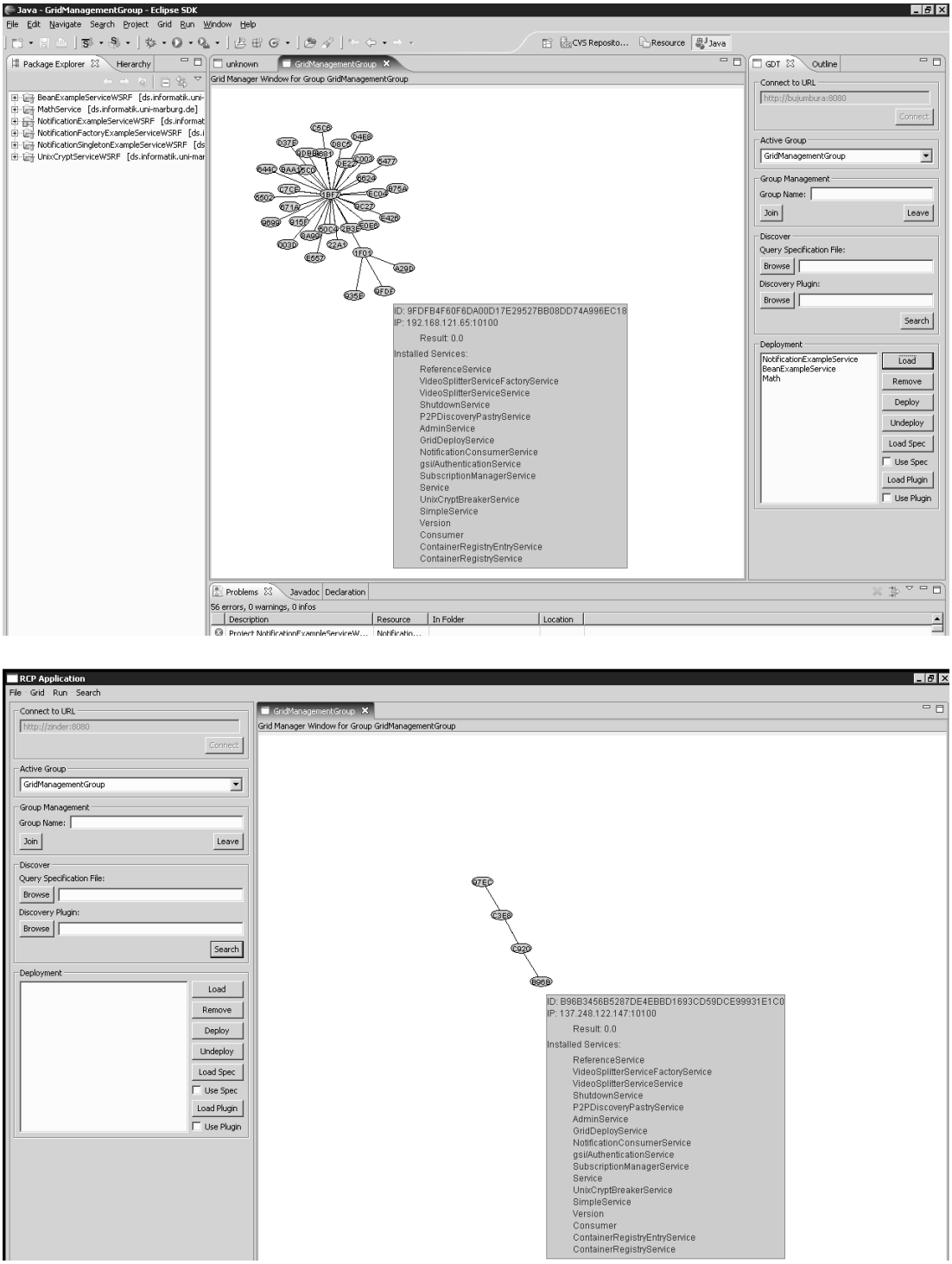


Figure 5.20: Grid management tools with Graphical Grid view integrated into the Eclipse workbench (top) and as a stand alone RCP application (bottom).

the HDS service on a Grid node of the ad hoc Grid or performs the deployment operation automatically on every node that is selected in the graphical discovery view for the ad hoc Grid environment. Effectively, the user is enabled to turn an application with a few steps into a deployable Grid service and instantaneously perform the deployment based on the selection of target nodes in a graphical view of the Grid. The same deployment operations are also implemented for the Grid process execution component and the corresponding process management service. These actions enable users to directly deploy processes contained in a BPR through the process deployment service. In this way, the GDT integrates specification and visual modeling of a Grid process with packaging for deployment and actual deployment into the Grid environment.

5.4 Summary

In this chapter, the prototypical implementation of the Marburg ad hoc Grid Environment (MAGE) was described. The MAGE implementation is based on the Globus Toolkit 4 as a basic WSRF compliant Grid middleware implementation. As first components, the implementation of the discovery and communication component of the system are described. In its current implementation, the system is based on the FreePastry P2P framework. After the discovery and communication components, implementation details for the Hot Deployment Service were presented, which enables the flexible ad hoc operation of the Grid environment by exposing component deployment as a higher level application functionality.

Closely coupled to the hot deployment functionality are the service isolation capabilities of the system, shielding the Grid service instances of different users against each other and protecting the underlying Grid node from dynamically deployed services. The implementation strategies both for the isolation of pure Java services and for fine grained isolation of Grid services that encapsulate native legacy components were described.

The description of the runtime components of the MAGE system were concluded by implementation details of the data handling component as well as the Grid process execution engine that are part of the MAGE system. The data handling component integrates middleware functions for efficient and flexible transmission of large amounts of binary data over different protocols while offering application developers an easy to use programming interface. The Grid process execution engine is an extension of the ActiveBPEL engine that has been altered to enable interaction with Grid services and to provide an additional process activity, the GridForEach construct. Grid process specifications are based on the BPEL standard.

In the second part of this chapter, the Grid Development Tools (GDT) implementation was described. The GDT project realizes a set of plug-ins for the Eclipse platform. A first component realizes a model driven Grid service devel-

opment tool that can use input from an UML modeling tool or annotated Java classes to automatically generate binding code to a Grid middleware. This initial component of the GDT project has been designed to allow easy implementation of generators for different service-oriented Grid middleware systems. Users are supported by allowing remote debugging of Grid services in a MAGE environment. Another component of the GDT environment is a visual Grid process editor that enables synchronous collaborative work on a shared Grid process model. All components of the GDT environment, including management components for Grid node and service discovery can be used in a stand alone application in addition to integrating them in the Eclipse development workbench.

In the next chapter, a quantitative as well as qualitative evaluation of the MAGE implementation is discussed.

6

Evaluation

6.1 Introduction

An evaluation of the overall design and the prototypical implementation of the ad hoc Grid middleware described in this work is presented in this chapter. This chapter is divided into two parts. First, a number of measurements and experimental results are presented that give some information about the performance of the key components of the ad hoc Grid middleware. These quantitative observations are followed by a qualitative discussion of the components introduced in chapters 4 and 5 describing their design and implementation, respectively. The main goal of the ad hoc Grid is to improve the usability of Grid systems. Therefore, a qualitative improvement in flexibility and usability of the middleware is more important than a quantitative improvement.

6.2 Measurements

A number of measurements were performed to assess the operational behavior of the MAGE platform for various aspects of the ad hoc Grid. The experiments were chosen to identify the overhead incurred by the MAGE specific components and implementations in comparison to the behavior of a standard GT4 environment. The performance values also should give a hint to users of the MAGE platform about the performance to be expected by the various functions. This chapter evaluates the performance for the main P2P based functions of the MAGE platform (i.e. direct communication and node discovery) as well as hot deployment and the micro jailing approach for native code isolation. The processing overhead incurred by the FlexSwA bulk data handling component is analyzed as the last

component of the MAGE runtime system. Additionally, the performance of the most time critical part of the Grid Development Tools for Eclipse is discussed.

Throughout the tests, MAGE containers were hosted on identical PCs with Pentium D 3.0 GHz (dual core) CPUs, 3GB main memory and 250GB 7200 RPM Serial-ATA hard drives with 8 MB cache. All machines were connected through a 100MBit/s switched ethernet. Measurements were taken from a client application running within Eclipse 3.2. The client PC used throughout the tests was equipped with a Pentium 4 HT 3.0GHz CPU, 1GB of main memory and 120GB 7200 RPM parallel ATA hard drive. The nodes of the test Grid environment were either installed with Linux (CentOS 4.3) or Windows XP (SP2).

6.2.1 P2P Communication

Initially, experiments were conducted to examine the communication overhead incurred by the P2P communication infrastructure of the MAGE platform as opposed to standard Grid service calls transmitted using HTTP over TCP. A simple echo service was implemented as a test Grid service that simply returns a String passed as an invocation parameter. The service was generated using the GDT in Eclipse and then deployed into MAGE containers.

As a first test case, the client creates a WS-Resource and performs 100 service invocations on this WS-Resource consecutively. The payload used for the invocation, i.e. the number of characters in the String passed as a parameter to the Grid service and returned from the service as a result value, is increased between experiments, ranging from 1 character to 100.000 characters in powers of 10. The round-trip invocation time for each service invocation was acquired by calculating the difference of the time value returned by the high precision system timer before and after the invocation (obtained through the Java call `System.nanoTime()`). Experimental assessment of the timer resolution showed that time intervals of at least two microseconds could be measured using the `System.nanoTime()` timer.

The table in figure 6.1 shows the measured results from the first experiment. T_{avg} denotes the average invocation times, σ is the standard deviation of the measurements. A plot of the experimental data can be found in figure 6.2(a) for Linux and 6.2(b) Windows respectively. The P2P transport implementation showed equal performance characteristics in both Grid service containers running on the Windows and the Linux platforms. Results for the standard HTTP transport greatly varied between the two platforms. Up to 10 kilobytes of payload, the average time needed for a service call using the P2P implementation took on the average 2.2 times the time required for a regular service call using HTTP. For larger payloads, the invocation times reached equal values for both transport types. A first inspection of the measurements for the Windows platform suggests that the P2P transport performs better than the regular HTTP transport.

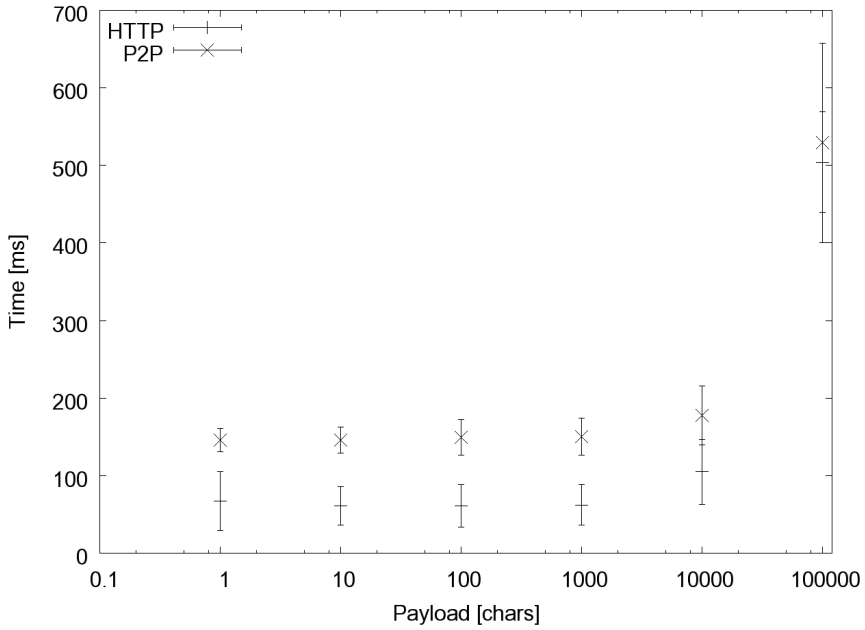
A second experiment was conducted, measuring 20 consecutive service invocations for various payload sizes between 0 and 10.000 characters increasing the size

	Windows XP			
	HTTP		P2P	
P [chars]	T_{avg} [ms]	σ [ms]	T_{avg} [ms]	σ [ms]
1	228.277	47.925	147.144	20.671
10	219.751	32.426	148.131	26.151
100	217.447	37.136	146.773	21.368
1000	45.537	22.139	149.866	22.616
10000	257.334	60.767	172.367	28.069
100000	612.218	85.579	498.038	85.326
Linux				
1	67.559	38.976	146.262	15.576
10	61.828	25.718	146.807	17.501
100	61.834	27.240	149.516	23.648
1000	62.911	26.384	150.318	24.702
10000	105.729	42.905	178.875	38.908
100000	504.598	65.432	529.561	129.381

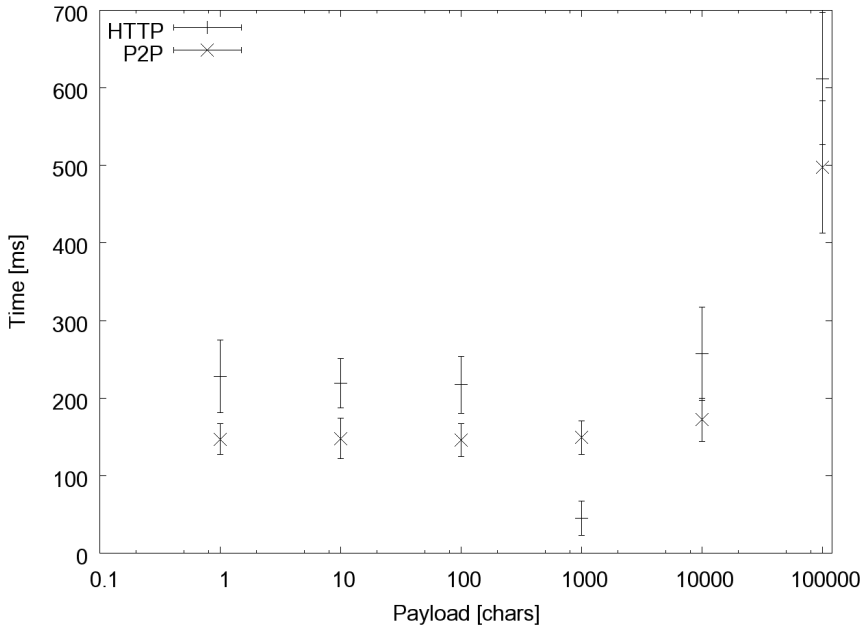
Figure 6.1: Average time T_a and standard deviation σ for invocation of a Grid service with various payloads P.

by 100 characters in every step. The results of this second experiment measuring the characteristics of the HTTP transport under Windows are shown in figure 6.3. The plot shown in figure 6.3 depicts typical results collected on several different Windows installations, from around 500 characters of payload up to 8000 characters. The container shows significantly higher performance than expected from performance values found below 500 characters of payload or above 8000 characters. This behavior of the standard HTTP transport is only found on the Windows platform, all tested Linux systems showed a linear increase in invocation time. Further investigation of the effects in the entire Grid container showed that the time required to perform the call `engine.invoke(msgContext))` - this call does the actual Grid service invocation - varied between the transports on the Windows platform. While identical request messages are passed to the engine in the message context, slightly different initialization of the message context is performed in both implementations. Below 500 characters of payload and above 8000 characters, the call was processed roughly 10 times slower than between the two values using the HTTP transport. A similar effect could not be observed on either the Linux platform or using the P2P transport under both operating systems. The effect seems to be related to the implementation of the JVM for Windows since the same implementation behaves differently on the Linux platform and no code directly related to the transport implementation is reached in the critical section.

Typical service invocation operations are expected to transfer an application



(a) Linux Grid service container.



(b) Windows Grid service container.

Figure 6.2: Time required for Grid service invocation.

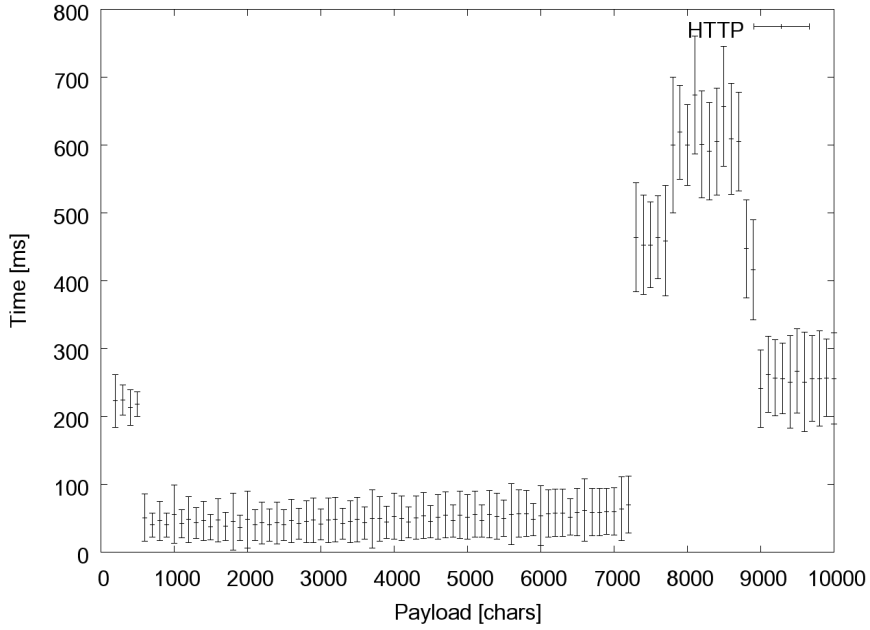


Figure 6.3: Average time for service invocation of a Grid service hosted on the Windows platform.

specific payload of under 10 kilobytes. The use of other transport protocols is highly recommended for larger binary transmissions. As a conclusion, the standard HTTP transport performed about 2 times better than the P2P transport in the most important payload region up to 10 kilobytes under Linux and roughly 3 times better under Windows. The total round-trip time for a service invocation using the P2P transport was about 150 ms using the P2P transport in contrast to 60-70 ms using the HTTP transport. Given that the implementation of the P2P transport has not been optimized for performance and should only be used when direct communication is impossible due to network partitions, it poses an acceptable overhead for service invocation, especially since the typical computation performed by a Grid service upon invocation lasts for several minutes up to hours or even days, in which case the increase under 100 ms poses a negligible overhead.

The actual time required for the transmission of SOAP messages through the P2P network is governed by the performance of the routing algorithms in the P2P network and subject to analysis by the developers of the P2P infrastructure.

6.2.2 Hot Deployment

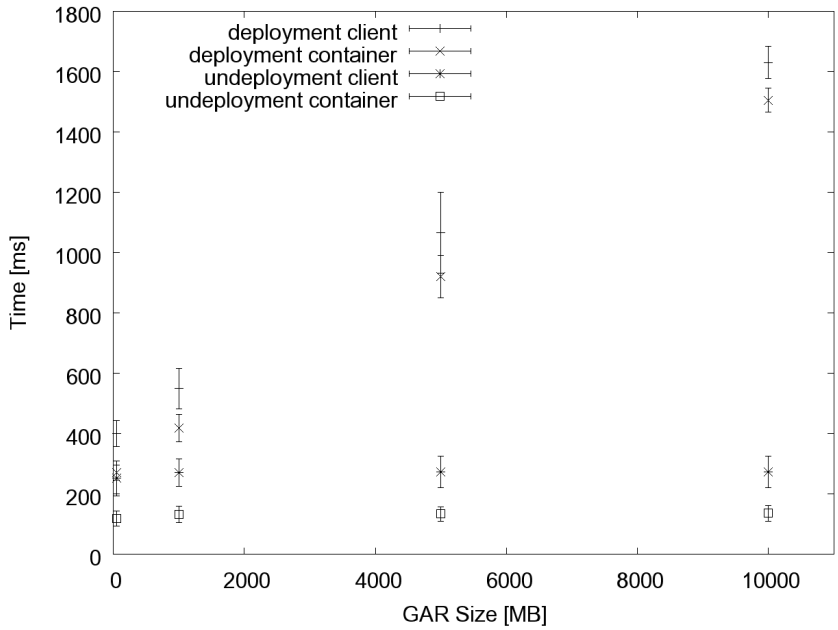
In a second series of experiments, the performance of the hot deployment service was tested. For this purpose, different Grid service archives with sizes ranging

	Linux							
	Deployment				Undeployment			
	Client		Container		Client		Container	
	T [ms]	σ [ms]	T [ms]	σ [ms]	T [ms]	σ [ms]	T [ms]	σ [ms]
50k	400.37	44.42	271.15	25.20	252.02	58.54	118.18	25.90
1M	549.63	68.07	418.21	46.17	270.98	46.42	131.11	27.88
5M	1066.30	135.54	921.08	71.71	272.14	52.15	133.46	23.81
10M	1630.33	53.75	1505.11	40.67	273.17	53.87	136.19	26.26
	Windows							
50k	772.54	59.04	323.84	18.45	492.60	33.45	64.57	23.38
1M	837.65	77.68	366.35	5.92	515.38	64.50	73.52	4.90
5M	979.31	71.07	512.43	17.29	538.78	60.99	75.65	6.73
10M	1141.78	64.77	692.96	5.90	493.74	47.43	74.23	9.81

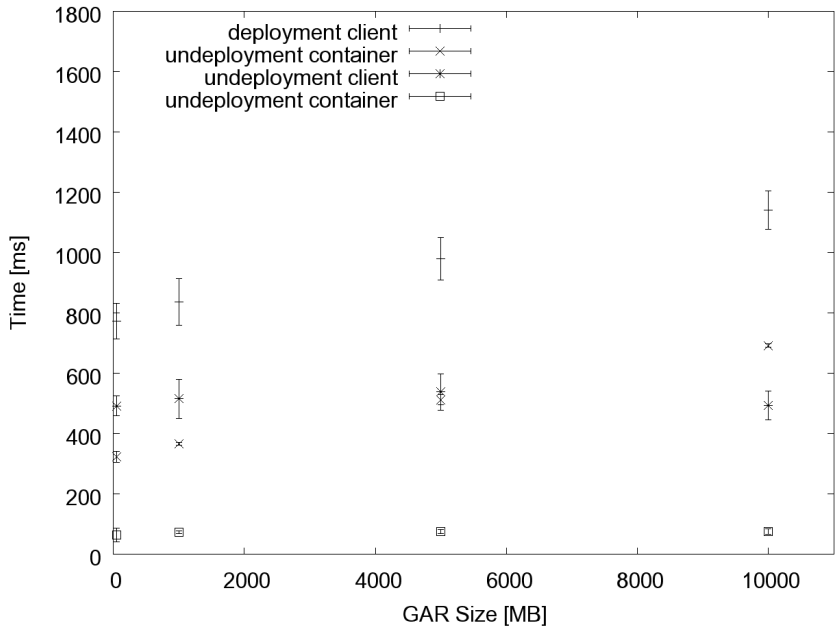
Figure 6.4: Time required for deployment and undeployment of Grid services of various size.

from 50 kilobytes to 1, 5 and 10 megabytes were generated. During the test, a series of 50 deployment and undeployment operations were performed on both MAGE containers used in the previous communication test. The tests used the method to deploy a GAR from the container file system in order to measure the raw deployment time excluding additional overhead for the transmission of the GAR file to the target node. Two values were collected for the individual tests: the overall time required for the deployment operation from the client perspective (i.e. the time difference before and after the synchronous invocation of the `deploy`-operation of the HDS) and the time required to perform deployment within the MAGE container (i.e. the time required to unpack the GAR, register all components with the platform to make the service accessible). All HDS invocations were performed using the HTTP transport. The results of the experiment are collected in figure 6.4. A graphical representation of the deployment timings is shown in figure 6.5.

The time required for deployment grows with the GAR size almost linearly. The deployment times measured within the Grid service container are the most significant measurements, since they do not contain the varying overhead for the transmission of SOAP messages. The performance values measured in the client also reflect the previously described anomaly of the HTTP transport on the Windows platform. Deployment messages contain only the reference to the GAR in the container file system, therefore, they are in the class of messages with a payload below 500 characters. This fact leads to a substantial invocation overhead clearly visible in figure 6.5(b). For a typical Grid service implementation under 1 megabyte of archive size, a client is able to instantiate a service around 600 ms (Linux) or 850 ms (Windows) after invocation of the deployment operation.



(a) Linux Grid service container.



(b) Windows Grid service container.

Figure 6.5: Time required for Grid service hot deployment from container filesystem.

Nodes	15	20	25	30	34
T_{avg} [ms]	468.74	800.14	1097.15	1198.00	934.98
σ [ms]	129.37	145.06	143.42	141.24	132.55

Figure 6.6: Average time and standard deviation of the discovery experiments.

The undeployment operation does only require the HDS to remove the registration of the service implementation from the Grid service container and deletion of the service code libraries and configuration artifacts from the file system. The operation is performed by the HDS using constant time. Again, the communication overhead due to the limited message size is significant on the Windows platform.

In contrast to hot deployment, the deployment operation provided by GT4 requires to stop the container, call an Ant script for local deployment and restart of the container (taking around 15 seconds on either platform), call an Ant build script that deploys the service into the GT4 distribution on the local machine and then copies the entire distribution into the Grid service container (taking over 5 seconds without previously deployed services) and restart of the container, again requiring around 15 seconds. Even assuming automation of all standard deployment steps, the overall operation requires over 30 seconds until a service can be instantiated while the more flexible hot deployment operation requires under 1 second to make the service available.

6.2.3 Discovery

The scalability of the P2P discovery component is mainly determined by the scalability limits of the underlying P2P infrastructure. To gather a feeling for the overall performance of the MAGE discovery component, the following experiment was conducted: Discovery queries were issued from a single node in the network for a list of all available nodes. Every 100 discovery queries, the number of total nodes was increased by 5. This increase happened by simply starting the nodes and allowing them to automatically join the same Grid group. Experiments were conducted on a switched 100Mbit/s Ethernet. The experiment started with 15 initial nodes in the network, the total number of nodes was increased up to 34.

The results of the experiment are depicted in table 6.6. The increase in time required between the 15 and 20 nodes can be attributed to the fact that the underlying Scribe system introduced another hierarchy in the application layer any-cast tree, that it constructed for the increased number of nodes. Performance and scalability of the MAGE discovery component is therefore directly correlated to the performance of the Scribe system. For more detail regarding Scribe and its performance, the reader is referred to [37].

6.2.4 Security

Isolation of pure Java service implementations is realized using a changed class loader delegation scheme. Since classes also need to be loaded by the standard implementation and the disposable and secure class loaders used in MAGE follow the implementation pattern of the regular class loaders, they do not impose an additional cost for loading of the classes. This security scheme also does not influence regular and repeated service invocations.

A significant overhead is created by the isolation scheme for native code fractions of a service, described in section 5.2.5. In this scheme, execution of the native code fractions of a service are performed in a process space separate from the JVM and sand-boxed by means of the underlying operating system. The solution is specifically aimed at isolation of native code that gets called from the Java service implementation via the JNI. Two main factors govern the performance of this solution for individual calls to native methods: The overhead imposed by the native sandbox of the operating system and the overhead imposed by the process dislocation technique that requires local inter process communication. Cost analysis for native code sandboxing is subject to evaluation by the creators of the various techniques offering a solution in this area.

The following experiment was conducted to assess the overhead of transparent process dislocation for native code used in a Java service implementation through the JNI. A Java test class was created that uses a native method implemented using C. The absolute time before and after an invocation of the native method in the Java class was measured in a loop of 500 invocations. In the first case, the original native library compiled using gcc 4.0.1 was used. Then, the native library was replaced by a transparent proxy library that used RPC communication to perform method invocation on the original library in a process separate from the JVM. Again, the runtime of the native method invocation in the Java class was measured.

The regular call took 3 microseconds on the average while the call using separate process spaces took 566 microseconds. This increase in time required to perform a native call by a factor of 182 is only acceptable if relatively few native calls are required from the Java code into native libraries. This situation is often met when a Grid service implementation should provide a front end to functionality provided by native libraries that are the result of very long development processes and can therefore hardly be re-implemented or ported. In this case methods provided by the library are used on a macroscopic level and the service will typically spend substantial amounts of its runtime in the native code alone instead of requiring many native method invocations in the Java implementation.

A benefit of the micro jailing technology lies in the small memory overhead created by the solution. The native process manager and all RPC related components of the MAGE system require less than 1 megabyte of main memory. This is a small overhead compared to process isolation by use of dedicated Grid service

	Linux									
References	1	2	3	4	5	6	7	8	9	10
T_{avg} [ms]	1.15	1.16	1.42	1.79	2.16	2.55	5.96	6.30	4.26	4.75
σ [ms]	0.69	0.62	0.53	0.48	0.56	0.64	0.55	1.12	0.74	0.89
	Windows									
T_{avg} [ms]	2.01	1.64	2.20	1.95	3.54	4.23	7.00	9.39	3.93	7.03
σ [ms]	0.89	0.54	0.46	0,95	0,59	0,53	0,80	10,87	0,50	1,30

Figure 6.7: Time spent by the Grid service container in the FlexSwA reference handler.

container instances for different services and users. Using these dedicated Grid service containers, a complete JVM must be instantiated, requiring at least 20 megabytes of memory.

6.2.5 Data Handling

The performance overhead by the data handling facilities of MAGE is mainly created by the reference handler that is configured into the request processing chain of the Grid service container. Performance of the binary data transmission is strongly dependent on the concrete transport protocol and comparable to the performance that can be reached by using the protocols in an application specific fashion. The main benefit of using the MAGE data handling facilities over custom application specific implementations lies in the reduction of code required in the application logic to perform flexible bulk data handling.

The reference handler of the MAGE platform is configured in the service specific request processing chain. Therefore, no overhead is created for services that do not use FlexSwA. To assess the time required for reference processing, a service invocation containing a FlexSwA reference was performed 100 times consecutively. Table 6.7 shows the average time T_{avg} spent by the Grid service container in the reference handler and the standard deviation σ of the measurements.

Other application specific implementations for bulk data transmission through external protocols need to process similar reference structures in their application logic. They can be expected to incur similar computational overhead on the invocation performance for a Grid service.

To investigate the performance of the post invocation parameter transmission (PIPT) supported by the FlexSwA component of MAGE, the overall processing time for 50 double values was measured. This experiment assumes that the service consumes parameters and processes them as soon as a parameter is available from the client. This pattern is often found in stream processing algorithms and cryptography applications. The overall time is the time needed to produce, transfer and process the parameters. Production and processing time were varied

	Standard			PIPT		
	$T_p = 1s$	$T_p = 3s$	$T_p = 5s$	$T_p = 1s$	$T_p = 3s$	$T_p = 5s$
$T_g = 1s$	100.2217	200.1592	300.2545	51.1484	151.1079	251.0996
$T_g = 3s$	200.1929	300.1747	400.2118	150.9913	153.1300	253.0937
$T_g = 5s$	300.2715	400.9557	500.1664	250.9778	252.9932	255.0877

Figure 6.8: Performance of regular parameter transmission and post invocation parameter transmission.

(1s, 3s, 5s per parameter), each experiment was repeated 10 times for every combination of generation and processing times.

Using traditional parameter transmission, the client produces the parameters, puts them in an array and sends the array to the remote Grid service. The Grid service iterates over the array and processes each parameter. Hence, parameter production, transmission and processing take place sequentially. In contrast, using PIPT leads to an overlapping of parameter production, transmission, and processing.

Figure 6.8 shows the overall processing time in seconds for traditional parameter transmission as well as parameter transmission using PIPT, under different parameter production T_g and parameter processing T_p times. The overall processing time T for n parameters follows the expected correlations: $T = nT_g + T_{SOAP} + nT_p$ for traditional parameter transmission and $T = max(nT_g, nT_p) + min(T_g, T_p) + T_{PIPT}$ for PIPT, where T_{SOAP} is the time required for a SOAP call and T_{PIPT} is the overhead incurred by continuous parameter transmission (including the initial SOAP call to trigger service execution). The communication overhead created by the SOAP message for the service invocation or the continuous transmission of parameters lies in the order of 100 ms. This experiment with relatively limited runtime on the service side indicates the impact of the communication overhead found in the P2P communication infrastructure.

6.2.6 Grid Development Tools

The most time critical element of the GDT is the project builder that gets invoked every time the user changes a resource in the Eclipse project that is part of a Grid service and under control of the GDT. The builder is invoked in a background thread after the Java builder of the Eclipse JDT feature has been invoked on the set of changes in the project. A run of the builder has two fundamental phases: first, identification of Grid service model sources and changes to those sources, then, the (re)-generation of service artifacts related to the changes. To test the performance of the builder, annotated classes were created containing either 5 or 25 methods or attributes tagged as `GridMethod` and `GridAttribute`. Then, the GDT builder was invoked 10 times after the start of the Eclipse workbench, each time the annotated class was touched and saved to force the GDT builder to run.

	Repeated Invocation			First Invocation	
	T_{avg} [ms]	T_{max} [ms]	σ [ms]	T_{avg} [ms]	T_{max} [ms]
5 Methods	666.25	1132.56	114.57	2022.33	3932.26
25 Methods	765.68	1232.69	123.95	2581.94	3076.47
5 Attributes	789.25	1265.52	116.77	2807.52	2980.52
25 Attributes	1013.34	1539.37	233.66	3232.47	3045.05

Figure 6.9: Performance of the GDT builder for different numbers of attributes and methods in an annotated class.

Figure 6.9 shows the average time T_{avg}, maximal time T_{max} and the standard deviation of the measurements σ observed during the experiments. Each set of experiments was repeated 5 times to gather performance data for the GDT builder including the first invocation that requires the Eclipse platform to load and instantiate the classes of the GDT plug-in. The tests were conducted on an IBM T41p notebook with a 1.7 GHz Pentium M CPU, 1 GB of main memory and a 60GB 7200 RPM parallel ATA hard drive running under Windows XP (SP2). Experiences with test applications built on top of the MAGE platform show that Grid services usually contained less than 25 distinct methods or attributes. After initialization of the Eclipse workbench, even the worst case performance requiring 1.5 seconds to perform a GDT build as a reaction to a manual save operation of the annotated class provide acceptable performance values to the user. Even more so, since the workbench is still usable by the user due to the background execution of the build.

The performance of the Grid Development Tools in headless mode or in an Ant build file is typically non-critical to the user. The automated build system performs nightly builds and tests of the MAGE platform and the GDT on a P4 HT 3.0 GHz under Linux. These automated tests were used to collect performance data for the GDT Ant task. Throughout the automated tests, various Grid services are built using the custom Ant task. These builds require initialization of a new Eclipse workspace, creation of Eclipse projects within the workspace for the individual services, import of annotated Java classes and generation, compilation and packaging of the Grid service implementation. These operations constantly require 35 seconds for the first invocation of the GDT task in the Ant build script. Subsequent invocations finish in under 20 seconds. This performance increase can be explained by a slight overhead required to create and initialize a new Eclipse workspace for the first service. Also, after the first invocation, the JVM and Eclipse classes are cached by the operating system, leading to a performance gain in subsequent invocations.

6.3 Qualitative Evaluation

While the previously described measurements give some information about the performance overhead connected with different components of the MAGE implementation, the main goal of an ad hoc Grid environment is an improvement in the usability of Grid technology for developers and users of Grid based applications. In this chapter, some qualitative improvements gained by the ad hoc Grid are discussed.

In general, the design of the ad hoc Grid middleware presented herein strictly follows the principles of service orientation and modularity. All interfaces of the basic components are exposed as Grid services to allow higher level applications access to those components. This allows application developers to design their applications with access to those key components and principles of the ad hoc Grid. Furthermore, the strongest possible modularization of the components allows developers in a concrete implementation of the ad hoc Grid middleware like MAGE to develop components with a great level of separation of concerns. It also opens the possibility to exchange certain implementations without affecting other components of the middleware. One example of such a component that might to be exchanged is the underlying P2P infrastructure used for node communication and node and service discovery. Different P2P protocols exhibit different performance characteristics and strengths and weaknesses in different environments that make them a primary subject of change in order to adapt the entire system to a concrete environment.

6.3.1 Communication and Discovery

An infrastructure supporting communication and node and service discovery that requires only limited manual configuration overhead has been identified as a basic requirement for an ad hoc Grid environment in section 2.3. The MAGE system includes such an infrastructure as a core component. The communication and discovery components are implemented using strong modularization that allows to replace the underlying P2P infrastructure. Currently, the FreePastry P2P middleware provided sufficient performance and functionality even though it presented some limitations. A common problem of P2P networks is the bootstrap problem, the act of joining the P2P network usually requires the address of at least one node in the network. FreePastry solves this problem by requiring the user to specify a bootstrap peer. MAGE provides a highly configurable bootstrap library that can either use predefined rendezvous peers or automatically identify bootstrap peers using broadcasts in a local network. The FreePastry middleware can currently not cope with network partitions like Firewalls and NAT routers. Support for tunneling through NAT routers is a feature under development for FreePastry. The measurements presented previously in this chapter give an idea of the costs connected with a single Grid service invocation using the direct

HTTP and the P2P transport components of the MAGE middleware. Compared to the expected long runtime of a Grid service, both implementations provide reasonable performance overheads.

From a user perspective, the P2P infrastructure enables a node to join a Grid environment by simply starting the MAGE software. The user just has to provide the group membership certificate in order to allow the node to interact with other nodes in the group. If there is no other MAGE node within range of multicast messages in the local network, the user needs to specify a rendezvous peer for the first contact with the MAGE network. Subsequent attempts to join the Grid network as the first node in a local network are supported by building a list of possible bootstrap peers during interaction with the MAGE network. Another benefit of the realization of the discovery component using a P2P framework is the fact that the discovery service is provided collaboratively by all nodes in the network. There is no need for a single party within the user group or virtual organization to install and maintain a dedicated directory service. The P2P infrastructure enables operation of an ad hoc Grid with very limited configuration overhead compared to the amount of manual configuration required by more traditional Grid middleware implementations such as GT4. The monitoring and discovery service of GT4 requires manual definition of a tree of index servers. A similar tree is constructed by the P2P infrastructure automatically in the MAGE middleware.

6.3.2 Hot Deployment

Hot deployment of Grid services is one of the most influential features of the ad hoc Grid. In a classical Grid environment, applications and services are assumed to be manually installed by an administrator or the Grid service provider in general. These applications and services are of rather static nature. Most Grid services in the traditional middleware systems are designed as service wrappers for certain infrastructural components also found in Grid middleware that is not service-oriented. As an example, consider RFT as the Grid service-oriented control interface for other underlying data transport protocols such as GridFTP or the ManagedJobFactory service of WS-GRAM that exposes the local batch queues of a cluster through a Grid service interface and creates WS-Resources to represent jobs in the queue. This view on service orientation does not represent services as the primary elements of Grid applications. Web service standards are rather seen as standardized communication protocols. Also, Grid services are usually not seen as the primary Grid application elements if WS-Resources are only used to represent jobs in a cluster queue.

In contrast, the ad hoc Grid offers the capability to deploy new Grid services into a running container and considers services as the primary elements used to build more complex Grid applications. The computational power of a node in an ad hoc Grid is directly tapped by the service implementation rather than

just using a WS-Resource to represent the execution state of a script or program enqueued in a cluster queue. This view on Grid services much better reflects the nature of an ad hoc Grid system, which is expected to be built from a large number of desktop PCs together with a few high performance resources such as computational clusters. It also changes the kind of service provided by the resource owners. The physical resource such as computational power is offered to other participants in the ad hoc Grid. Those users can either deploy application specific services for their own use or offer services on the raw underlying resources of the resource owners. Furthermore, administration of a Grid node is substantially simplified since fine grained control may be imposed over individual services deployed in a Grid service container. The preparation of the application environment in the ad hoc Grid can become part of the application itself.

Hot deployment of services is a feature of the ad hoc Grid used extensively in the medical as well es the media analysis sample application presented in section 2.2. In both cases, flexible change of the algorithms and building blocks used in a more complex application workflow is a primary goal and research interest. It is undesirable to manually install a new service on every node participating in the Grid environment. As a further example, consider the need to provide a service with uncertain and changing resource demands. Hot deployment of a Grid service is a feature that can easily be used to incorporate more raw resources into a computational Grid that is used to provide a Grid or even Web service answering different levels of demand.

6.3.3 Security

Hot deployment of services and the envisioned concurrent usage of a Grid node by different users drive the demand for intra-node security. The need for strong separation between different users has partially been considered in more traditional Grid environments by employing standard orange book security schemes, that hide, for example, process information of one user of a cluster system from other users. However, they often require strict allocation of a computational node to a single user and thus can easily lead to ineffective use of available hardware resources. This fact is especially important in application scenarios where service provisioning should be scaled to the demand e.g. to answer peak loads economically.

The ad hoc Grid middleware design mandates the implementation of provisions for isolation of service instances of different users against each other. Additionally, isolation of the Grid service hosting node against the service instance is required and answered by the isolation scheme defined for the ad hoc Grid middleware and implemented in MAGE. Where standard GT4 installations require trust in the implementation of a service or the applications that should be installed on a node, the ad hoc Grid defines secure sandboxes to be used for the deployment of services and applications. This isolation scheme gives a means

of protection of the underlying node from the services and software deployed to that node. The node owner can define the access policy to be applied to each service, retaining fine grained control over the level of access a service gains to the local machine.

Many scientific applications require the use of legacy code implemented in a native language other than the typical interpreted languages used for Grid service implementations. The need for sandboxing and restriction of access to the underlying platform by native code is clearly mandated by the ad hoc Grid middleware. MAGE provides an implementation of the service isolation component that answers this need for isolation and access restrictions of native code. This implementation is also highly modular and can make use of different sandboxing technologies provided by the underlying operating system (e.g. Systrace system call interposition, BSD Jails, Xen para-virtualization).

For the final vision of an ad hoc Grid, the protection of a Grid users' code and data from the underlying platform is required. Such a protection could be realized using technology developed by the trusted computing platform alliance (TCPA). Unfortunately, the required TCPA hardware is not yet available or far enough advanced to truly support the implementation of a Grid environment that can meet this demand. Such a middleware should be securely verified during runtime and offer the guarantee to a user (i.e. service deployer and input data owner) that protection of data and custom code against a node owner is ensured.

Access control to the services provided by a Grid node and access of nodes to an ad hoc Grid environment in general are granted based on a group concept. The basic mechanisms to control access to a service based on certificates are already provided by standard service-oriented Grid middleware. Consequently, these components are re-used and extended in the access control mechanisms of the MAGE implementation.

6.3.4 Data Handling

Efficient and easy to use data handling mechanisms are a requirement for many Grid applications. The requirement for a data handling component in the ad hoc Grid can be assumed to be somewhat different from the general requirements for data handling in other large scale scientific Grid environments. There are dedicated data Grid developments that have to cope with the fast transport and storage of very large amounts of experimental data on a global scale. These installations require dedicated and extremely fast network connectivity at experimental sites and large buffering architectures to cope with an amount of experimental data exceeding several terabytes per second.

The data handling component of the ad hoc Grid middleware presented in section 4.2.6 was especially designed to ease data handling from a user's and programmer's perspective. It offers an easy to use and flexible data handling component for large amounts of binary data. In addition to allowing transmission

of attachments to a SOAP call over different binary protocols, it offers the ability to define transmission policies between a client and a Grid service that also allows for more efficient overlapping of data transmission and service execution.

The data handling component uses a modular design that allows to implement many data transmission patterns and handlers for various protocols. The FlexSwA infrastructure anticipates the use of different protocols that can be changed by configuration during runtime. Protocol negotiation support in the FelxSwA infrastructure also allows the same client to use different means of data transmission with different Grid nodes or Grid service instances based on the data transmission policy that the node owner or service provider has defined together with the preferred behavior defined for the client.

The implementation of the FlexSwA component for the MAGE middleware integrates efficient support for the handling of large amounts of binary data in SOAP calls, which is not supported by the standard GT4 implementation that forms the basis of MAGE. GT4 defines the reliable file transfer (RFT) component to handle the transmission of large amounts of binary data via GridFTP. RFT offers a Grid service interface that takes scripts as an input that define the data transmission between different nodes. Applications are required to contain a relatively large amount of client code or static scripts that control the transmission of binary data. In contrast, the FlexSwA component allows to directly reference or virtually embed large amounts of binary data in SOAP calls while embedding large portions of standard data handling code in the middleware that can be selected by policy definitions rather than complicating Grid applications by incorporating data handling application logic.

This feature of the data handling component defined for the ad hoc Grid and implemented in the MAGE middleware form a fundamental building block to ease the development of Grid applications for application domain Grid experts that lack strong knowledge about Grid middleware. The data handling component enables a programming style that allows users to directly pass data references in SOAP messages while the platform handles actual transmission of the data. This also allows the platform to pass references to other Grid nodes for processing without actually creating the overhead for the transmission of large amounts of binary data. Reference passing also enables the easy application of a Grid process execution engine to construct more complex applications from basic Grid services as application building blocks. The FlexSwA infrastructure enables the Grid process execution engine to define the control flow of an application and pass data by reference between nodes by direct reference to the data rather than actually having to transmit the data through the central process execution engine.

6.3.5 Grid Process Execution

The process execution component of the ad hoc Grid middleware offers a fundamental means to define Grid applications from a high level perspective. The

idea of process execution as a composition of basic component services has been successfully applied in the business domain. The most common standard for Web service orchestration in the business domain is the business process execution language for Web services (BPEL). This standard was adopted for the Grid process execution component of the MAGE middleware.

At the time of development of the MAGE middleware, the current BPEL 1.1 standard and consequently all freely available implementations of process execution engines could not support the concurrent and parallel execution of service invocations that is commonly used in Grid applications. A basic requirement for this parallel execution support is the definition of partner links per BPEL scope, which will be introduced in the upcoming BPEL 2.0 standard. Even though the upcoming BPEL standard defines the necessary elements to model parallel and concurrent service invocations similar to the mechanisms defined for the ad hoc Grid by the GFE construct, most of the service discovery and scheduling logic must be explicitly defined in the application logic of the business process specification.

The Grid process execution support component of the MAGE middleware contains a standard implementation of a discovery and scheduling mechanism that can be configured to meet many application needs. Therefore, many standard application requirements can be met by simply configuring the existing solution with a limited set of parameters instead of explicit implementation of discovery and scheduling strategies in the abstract application logic of a Grid process.

A positive aspect of the adoption and extension of the BPEL standard is the broad acceptance and adoption of BPEL in the business domain. There is an increasing number of software tools that support the development and management of business processes based on the BPEL language. Many business application experts already developed strong knowledge in the definition of business applications based on the paradigm of service composition. It can be expected that the integration of Grid based applications in such an environment will find more adoption if the technologies used are well established and understood in the surrounding enterprise application environment.

6.3.6 Development and Management Support

Experiences gained during the development of Grid applications based on the GT4 middleware showed that even developers with experience in distributed application development struggled with a steep initial learning curve and the complexity of component development. Synchronizing the service implementation with the various artifacts required to deploy a Grid service into the service container proved to be an error prone and demanding task. Many of those errors could have easily been resolved if there was sufficient tool support automating the mapping between for example WSDL descriptions and service implementations, helping to keep both in sync. Consequently, service creation support is one of

the most important core components of the development and management tools of the ad hoc Grid middleware system presented in this work.

GDT offers a powerful set of tools that is integrated into Eclipse, one of the leading interactive development environments for Java applications. All tools have been designed with a focus on extensibility and adaptability to different service-oriented Grid middleware systems. On a conceptual level, this portability and adaptability to different service-oriented Grid middleware systems is aided by the separation of the platform specific model for a Grid application and components into an upper application oriented and a lower middleware specific model. Grid application development support often focuses on APIs abstracting from concrete middleware systems and custom build tools that help with the initial generation of certain elements of a component implementation. The GDT service creation component realizes round-trip engineering between a concrete Grid service implementation including all platform specific artifacts and higher level models of the component and more complex Grid applications. This synchronisation between generated classes and artifacts allow users to change not only the high level model source of a service but also the implementation with the result that the propagation of the changes into the model is proposed by the system. Using the core model of a service as the authoritative source, all elements of a service implementation can easily be synchronized.

The binding between application logic and higher level models of a Grid service and the automatic generation of service artifacts and implementation of the code that binds the application logic to the middleware platform leads to very fast development of Grid services. Observations showed that equally skilled groups of developers usually needed more than two weeks to confidently develop small Grid services using the existing middleware and development tools. Utilizing the GDT, developers were able to develop similar service in only a few minutes. The development of Grid services can further be aided by the integrated debugging support component. In a classical Grid environment, all debugging efforts relied on logging and explicit debug output. Interactive debuggers are very common yet unsupported in classical Grid environments. The GDT environment enables developers to attach to a Grid service container and inspect the execution state of the entire Grid engine including the Grid services. The GDT debugging components ensure that the correct source code version of the modules under inspection by the debugging component are available to the debugging infrastructure.

While the previously described Grid process execution component offers a mechanism for composition based creation of complex Grid applications, the same problems found in the component development process apply to this high level programming task. The GDT integrates synchronized (i.e. same time, different place) collaboration support for the creation of Grid processes, enabling application domain experts and middleware experts to interactively collaborate in the design and development of Grid applications. The ability to interactively consult with middleware experts eases the entrance into the Grid for non-Grid experts.

All management components such as the discovery and visualization component for the ad hoc Grid environment can be integrated in the standard Eclipse workbench. These components provide management functionality for the ad hoc Grid environment that can directly be integrated with the development tools. As an example, the developer can visually select a node from the ad hoc Grid environment and interactively attach the debugger to this node and directly deploy a Grid service to the node. The architecture of the Eclipse platform easily enables the re-use of all management components in a stand-alone application that can be used by Grid administrators to monitor and manage the ad hoc Grid environment. The same versatile applicability has been considered for the rest of the GDT components. Especially the model transformers and code generators represent strong development tools for automated build and test environments. Therefore, all components expose their functionality to a command line and scripting oriented environment by providing ANT and command line user interfaces in addition to the graphical Eclipse user interface integration.

6.4 Sample Application Scenarios

Three sample applications for a service-oriented ad hoc Grid environment have been initially presented in section 2.2 to give a motivation for the development of such a middleware system and to derive requirements for the framework. In the following, a brief discussion of the prototypical realization of these sample applications is given. Since the medical and media research applications are quite similar regarding their overall structure and requirements for the underlying middleware and the realization of the final application, only the development of the media analysis application will be discussed in this section. In addition, the realization of the engineering application using the tools and infrastructure of GDT and MAGE will be presented.

The development of the media analysis Grid application starts from a base implementation of the required application logic that is intended to run on a single node. The first use case for the application is a video cut detection application. The cut detection algorithm requires a two phase processing of the input data. First, a number of features are extracted from the video content. Second, these features are used to determine the cuts in the input video. Since feature extraction is the most time consuming part of the application, this step in the algorithm workflow should be distributed over different nodes in the Grid environment. While the original application code remains in a separate library, a simple core service implementation was created to implement the core feature extraction service using the GDT within the Eclipse workbench. The resulting `FeatureExtractor` service has one public method `extractFeatures` that takes a collection of Flex-SwA references to video segments as input and returns the corresponding feature lists as a collection of Flex-SwA references to the client.

```
package de.fb12.videoanalysis;

import java.io.File;
import java.util.Vector;

import de.fb12.flexswa.infrastructure.Reference;
import de.fb12.gdt.GridService;
import de.fb12.gdt.GridAttribute;
import de.fb12.gdt.GridMethod;

@GridService (
  name = "FeatureExtractor",
  namespace = "http://videoanalysis.fb12.de/FE",
  targetPackage = "de.fb12.videoanalysis",
  serviceStyle = "SSTYLE_SIMPLE",
  resourceStyle = "RSTYLE_MAGE")
public class FeatureExtractor
{

  @GridMethod public Vector<Reference>
    extractFeatures(Vector<Reference> refs)
  {
    ExtractFeature ef = new ExtractFeature();
    Vector<Reference> results = new Vector<Reference>();

    for(int i = 0; i < refs.size(); i++)
    {
      File f = refs.get(i).downloadFile();
      ef.extractFeature(f.getAbsolutePath());
      String fN = f.getName() + "_0.vd";

      results.add(new Reference(Reference.createName(fN)));
    }
    return results;
  }
}
```

Figure 6.10: Annotated class of the feature extraction service.

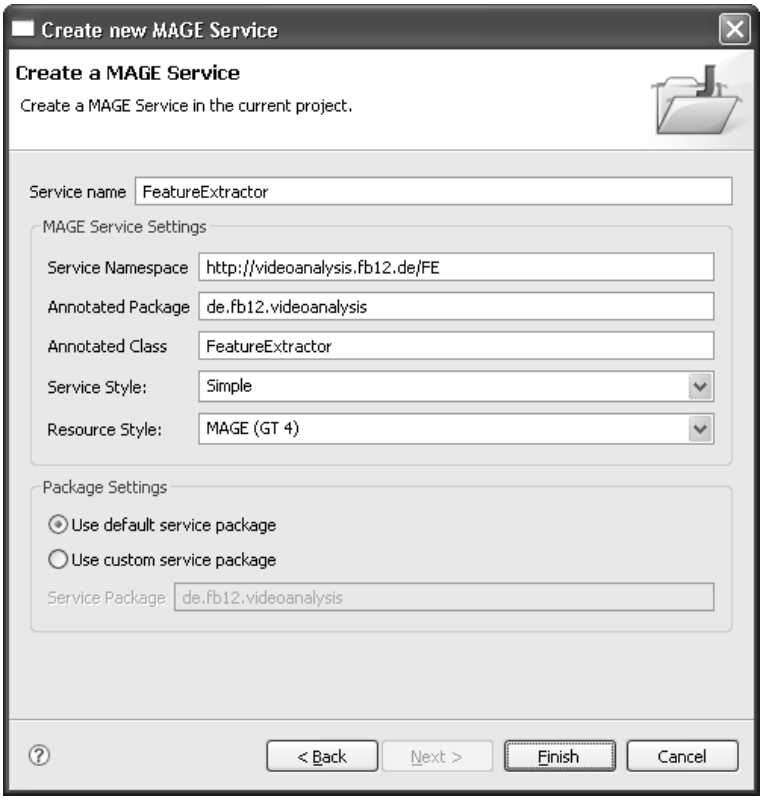


Figure 6.11: New service creation wizard with settings for the FeatureExtractor Grid service.

Figure 6.10 shows the entire annotated class carrying the feature extraction logic in the extractFeatures method that was manually created by the developer, the rest of the class was created as an implementation skeleton by the GDT new service wizard shown in Figure 6.11. After creation of the annotated service implementation stub with the Grid service wizard, the developer is required to add 9 lines of code manually. This code is required to receive the input data and make a call to the original feature extraction code in the existing video analysis library. Upon saving of the source file, the GDT builder automatically generates 5 new classes with 311 lines of code and 6 interface description and configuration files with 151 additional lines of text, that are required as an input to the existing tools for creating the final Grid service binding code and packaging of the service implementation. The developer then needs to invoke the last step in the integrated Grid service development process to build and package the distributable Grid service archive.

In addition to this feature extraction service, two other services are required for the distributed execution of the feature extraction, a video splitting service that can separate a single input video into smaller chunks for feature extraction and a feature merging service that combines the results from the individual

feature extraction steps. Again, the two services were created using the Eclipse workbench. Both contain roughly 20 to 30 lines of application logic carrying code and are marked as Grid services with the corresponding annotations. Both service implementations make use of the functions provided by the original media analysis library. The `VideoSplitter` and `FeatureMerger` service are automatically generated and packaged similar to the previously described feature extraction service. For the sample application, the developers decided to incorporate the final cut detection (i.e. feature interpretation) algorithm in the client of the overall Grid application instead of wrapping this algorithm as a Grid service as well, since the algorithm does not pose a performance problem.

After the creation of the component services, the developer can compose them into a higher level application workflow using the Grid process editor of the GDT. Figure 6.12 shows a screenshot of the feature extraction process in the collaborative process editor of the GDT. This process can contain steps to invoke the hot deployment service of MAGE to prepare the execution environment and distribute the feature extractor service to as many nodes in the Grid environment as possible. The application developer can also use the integrated management functionality in the Eclipse workbench to directly deploy the services to appropriate nodes in the ad hoc Grid environment. The `VideoSplitter`, `FeatureMerger` and `FeatureExtractor` services are combined using the `GridForEach` construct to perform distributed feature extraction. In addition to the straightforward feature extraction service implementation, three different services have been developed to conduct experiments with the Flex-SwA infrastructure and different data transmission policies. References to the services can easily be exchanged using the process editor and the process is then redeployed into the Grid environment that is formed by a collection of personal computers. All workstations join the same Grid group and can be used to perform feature extraction. The high level application process is again exposed as a Grid service. The developer implements a client to this service in the Eclipse workbench, and includes logic for the interpretation of the extracted features in the process client. For the prototypical realization of the application, a stand alone client was implemented that provides the video cut list as a file to the user. However, the application can easily be integrated into the Mediana workbench and provide a computational backend for the media workbench.

Developers can easily create and deploy additional feature extraction or video processing services in a similar manner and include them in the application workflow. The deployment descriptors and process specifications of derived and modified Grid processes are easy to export from the process editor and also dynamically deployable into the Grid environment. In the medical sleep analysis application scenario, the service-oriented ad hoc Grid middleware and development tools may be used in a very similar manner. Basic services may be flexibly combined to high level analysis processes on given data sets.

As a third sample application, the engineering process of collaborative devel-

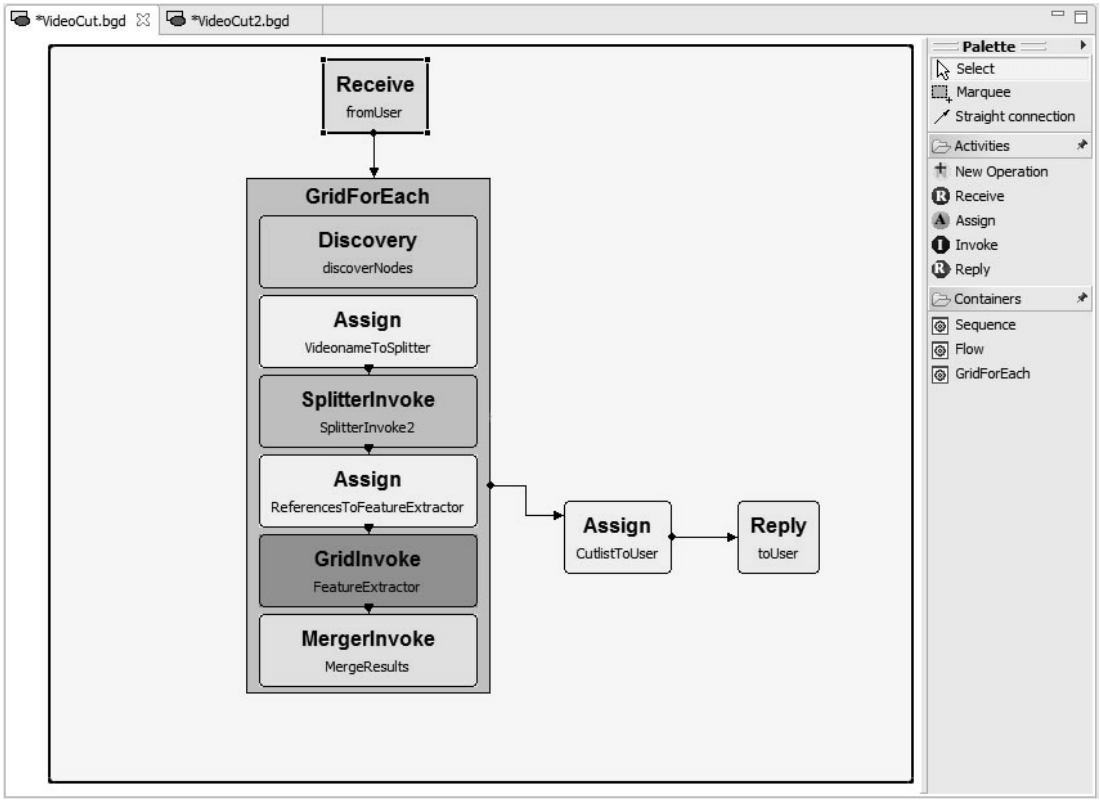


Figure 6.12: Feature extraction process for video cut detection in the collaborative Grid process editor of the GDT.

opment of a metal casting model is considered. From a software development point of view, the Grid relevant development cycle starts with a problem definition, expressed by the casting engineer and progresses through some iterations of model definition, simulation and refinement. The given problem definition is then modeled as an initial casting process model by a numerical simulation expert. Usually, this expert is located in another company because of the already mentioned lack of personnel or know-how in small and medium engineering enterprises. The numerical simulation expert periodically discusses the evolution of the initial model with the casting engineer during the design phase. Both experts have to combine their expertise to successfully define an accurate model for the casting process. To verify the accuracy of the resulting model, it typically is numerically simulated. The results must be reviewed and compared to knowledge about real casting processes held by the casting engineer. If this first simulation run does not match reality, the model needs to be calibrated and further model variants are created by the simulation expert and the casting engineer.

As a first step towards the optimization process, the casting engineer and numerical simulation expert join a collaborative process design session using the distributed process editor of the GDT. The distributed process editor allows users to join the collaboration and create, delete, connect and move basic activities concurrently. This interaction may be augmented by audio or videoconferencing giving the participants an on-line meeting room environment for their collaborative work. Figure 6.13 shows a snapshot of the distributed process editor with the minimal core process for the Grid application being worked on. The ability to concurrently edit the process model is a critical element in the overall development process since communication difficulties between the different experts can be quickly identified and solved. As stated before, they rely on cooperation to apply their combined expertise to design an application that can support the engineering process in a satisfactory manner. After finishing an initial sketch of the ideal application workflow from their domain perspective, they involve a Grid expert to help them adapt their process to the Grid environment and identify the necessary component services for their Grid process. The Grid expert will introduce infrastructural requirements such as service discovery and infrastructure management into the purely application oriented workflow of the domain experts.

As a result of this second step, a process for the Grid application and the specification of the required component services can be used by the Grid expert to implement or select the basic components. A basic skeleton implementation of the required services can be automatically generated using the GDT leading to fast availability of the component services. For the metal casting sample application, the following two services were identified and implemented:

The Distributed Polytop Service. This service is an implementation of the distributed polytop optimization algorithm [24] (DPA) which belongs to the class of direct search methods. It has its roots in the Complex algorithm [29], a predecessor of the Nelder-Mead Simplex [126]. The DPA was designed regarding

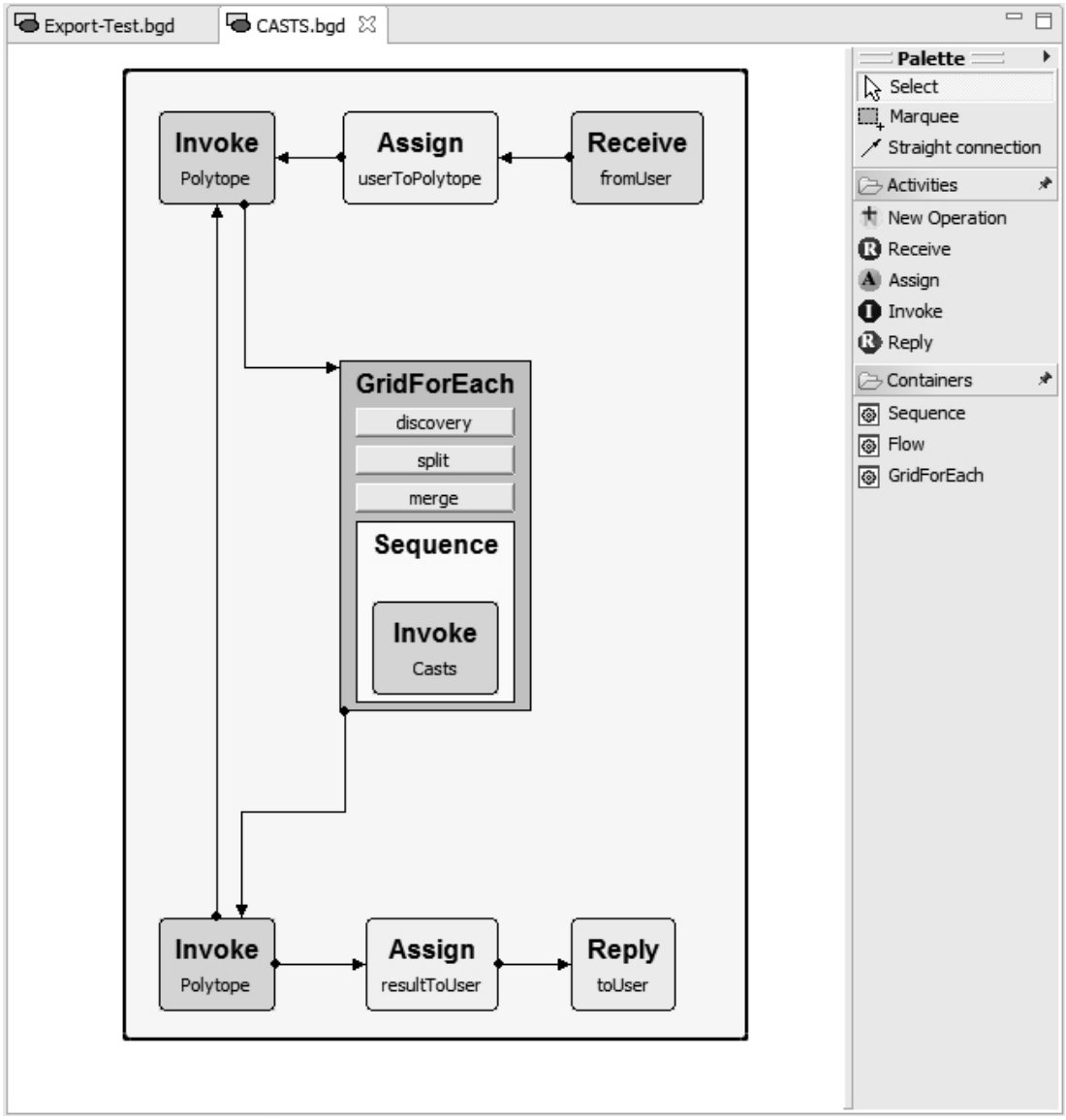


Figure 6.13: Screenshot of the graphical distributed process editor window displaying the metal casting process.

efficiency and scalability in distributed systems.

During its runtime, it requires an a priori unknown number of evaluations of both an objective function and corresponding constraint functions, in this case calculated by the metal casting simulation software CASTS [110]. The service has to save its state each time an evaluation request occurs, and it passes the data set which is to be evaluated to the process execution engine instead of directly invoking the simulation service. Considering these conditions, the service was implemented by utilizing the web service resource framework (WSRF), which allows the creation of stateful web service resources. Beside a service operation which allows a client to set necessary parameters needed by the polytop algorithm, the only Grid service operation `iterate(IterateRequest)` takes care of starting and restarting the algorithm at the appropriate position - according to its internal state and according to the input data inside the `IterateRequest` data structure. A resulting data set is returned immediately after invoking the operation, telling the process execution engine if further evaluations are needed or if the polytop algorithm reached a predefined stop condition.

The Casts Service. The main purpose of this service is to wrap the metal casting legacy software CASTS as a Grid service. However, the *Casts Service* does not only provide a service-wrapped version of CASTS, but it also takes care of the following operations: It is capable of modifying the input model of the casting process according to a set of parameters passed to the service. This parameter set is the input received from the distributed polytop algorithm.The service executes the CASTS legacy application on a number of different execution platforms. In this case, a 128 node cluster computer with two 64Bit AMD Opteron CPUs and 2GB main memory per node was utilized, leading the execution subsystem to incorporate the local resource manager Torque [47] and the scheduling system Maui [46]. The execution state of a cluster job is monitored and exposed by the Casts Service. The execution subsystem is highly modularized so that the service also works on single workstations without local queuing/scheduling. The service also provides functionality to evaluate the simulation result (which is done by CritCASTS, a legacy software system bundled with CASTS) and determining the objective function value as well as the constraint function values.

Utilization of WS-GRAM for running CASTS jobs in a Grid service wrapped command line was inappropriate due to the complexity of the internal tasks of the Casts Service. The chosen approach of a custom wrapper service exposes the necessary information and results much cleaner to the process execution engine. Using WS-GRAM would have required to create a sophisticated shell script as a CASTS wrapper and the inclusion of logic in the Grid application process for parsing the output of WS-GRAM, which truly belongs into the Casts Service.

The concurrent execution of many simulations has again been modeled using the GridForEach construct. The GFE construct neatly encapsulates details of the concurrent execution in the simulation tasks, keeping middleware complexity from the domain experts.

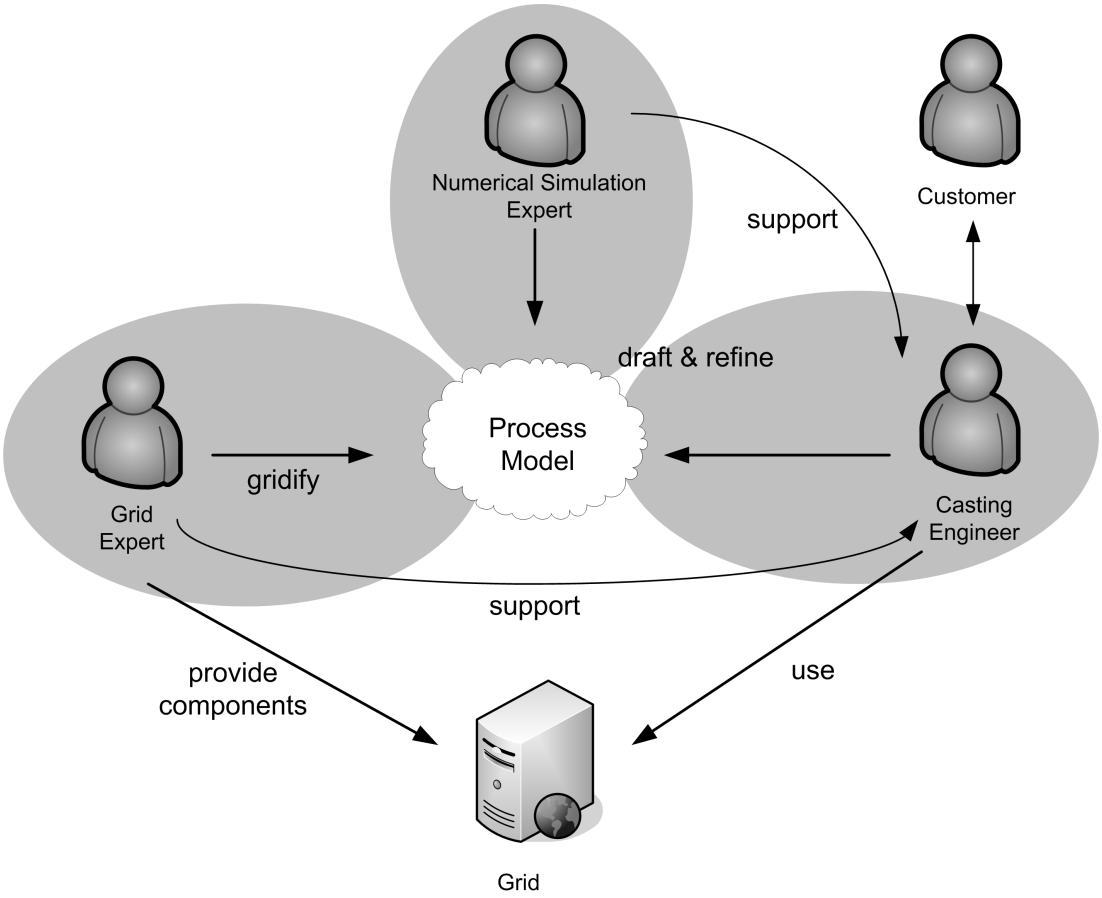


Figure 6.14: Overview of the collaborative process creation scenario for the metal casting application.

An overall view of this collaborative and distributed development scenario is shown in figure 6.14. The grey zones mark the network domains of the different experts, they are geographically distributed, and their collaboration takes place via the shared and synchronized process model. The distributed collaborative process editor allows them to synchronously edit the model and directly see the operations of other connected partners.

Finally, the resulting process can be deployed and executed in the service-oriented ad hoc Grid environment. The initial prototype of the application was tested in an application environment at the University of Siegen, Germany, that included a large AMD Opteron cluster system as computational resource for the execution of many instances of the CASTS services. Client computers were equipped with the MAGE middleware and integrated into the classical Grid environment also containing a large HPC resource. For more detail regarding the sample application scenario, the reader is referred to [84].

6.5 Summary

In this chapter, an evaluation of the overall design and the prototypical implementation of the ad hoc Grid middleware described in this thesis was presented. First, a number of measurements and experimental results were discussed that give information about the performance of the key components of the MAGE ad hoc Grid middleware. These quantitative observations are followed by a qualitative discussion of the components described in this thesis. This evaluation focused more on a qualitative than a quantitative evaluation, since the main goal of the ad hoc Grid is to improve the usability of Grid systems. In the last section of this chapter, the realization of the sample applications presented in section 2.2 using the MAGE middleware and the Grid Development Tools for Eclipse were described.

7

Conclusions

7.1 Summary

In this thesis, the design and implementation of an ad hoc Grid environment has been presented. The vision of the ad hoc Grid evolves from the service-oriented Grid towards a more robust, more flexible and more easily usable yet standards-compliant Grid. The current implementations of service-oriented Grid middleware systems still focus strongly on a traditional view of high performance computing, and realize a Grid environment as the combination of several cluster sites that are combined in a virtual organization network bridging the borders of the participating organizations. The ad hoc Grid idea propagated in this work goes a step further by including standards compliant Grid middleware components that can be integrated into such traditional service-oriented Grid environments, with more flexible components and usage patterns. This enables the construction of a combined environment of dedicated high performance computing resources with a transient personal Grid solution built from a number of personal computers.

As an introduction to the overall problem that needs to be solved by such a service-oriented ad hoc Grid middleware solution, three sample applications from medical research, media analysis and mechanical engineering were introduced. While the first two applications emphasize the use of the ad hoc Grid middleware to form an environment that collects computational resources from a network of personal computers, the last example from mechanical engineering has a stronger focus on a more traditional Grid view, combining large scale high performance computing resources. From these sample applications, a number of requirements were deduced regarding the runtime environment of a service-oriented ad hoc Grid environment as well as the development support components required to ease adoption of the Grid computing paradigm in the different applications.

For the runtime environment of the Grid system, these requirements include:

- Seamless communication even across barriers in the underlying network between different partners
- A discovery infrastructure that is easy to set up and maintain, flexible and robust against component failure
- The ability to dynamically deploy and use Grid services in the entire system without interfering with the services of other users of the system
- Strong protection of the underlying resources of the Grid environment from dynamically deployed Grid services as well as protection of the services and data of one user from potentially malicious other users of the system.
- The ability to model and execute complex Grid processes that combine the functionality of basic component services
- Flexible and easy to use data handling and transmission methods for large amounts of binary data especially in the context of Grid service compositions

The following requirements have been identified for a user friendly development environment for the service-oriented ad hoc Grid:

- Support for the creation of Grid services and higher level Grid processes/-workflows
- Separation of concerns between the application and the middleware domain and a high degree of abstraction from middleware specific questions for non-Grid experts.
- Support for synchronous collaboration of different physically separated experts
- Integration of Grid management components into the development environment to enable faster test of the application components

These requirements are reflected in the design and implementation of the Marburg ad hoc Grid Environment (MAGE) and the Grid Development Tools for Eclipse (GDT) in chapters 4 and 5. MAGE is an implementation of a WSRF compliant Grid middleware that embodies the ad hoc Grid features described in chapter 1 of this thesis. The GDT environment extends the Eclipse development environment to provide development support for the service-oriented Grid including traditional service-oriented Grid middleware systems as well as the ad hoc Grid runtime environment.

This thesis was concluded by a discussion of some quantitative penalties imposed by extending a service-oriented Grid middleware and the qualitative improvements provided as a result. The direct comparison of the performance indicators for different implementations need to be evaluated in the context of the robustness and flexibility gain that can be achieved. As an example, consider the increase in the average round trip SOAP service invocation time by a factor of 2 to 3 from 60 to 180ms for a representative service with respect to the parameter size that gets transmitted. Taking into account that the average runtime for a service is expected to be several minutes or hours, the increase by a tenth of a second is negligible. Additionally, an application using the P2P communication infrastructure will work in network environments where the standard HTTP transfer methods fail.

Other qualitative improvements in flexibility of the ad hoc Grid middleware such as the hot deployment service also improve performance values. The HDS enables service providers to make their services available in around 1 second, whereas the standard deployment solution takes around 35 seconds until a service is available. Such a performance is required when deployment of a service becomes part of the Grid application as in the ad hoc Grid. The overall concept of hot deployment as a fundamental primitive for the development of complex Grid applications is one of the major research contributions of this thesis, together with the general idea of developing a service-oriented ad hoc Grid environment first proposed in 2004 in several publications the thesis is based on. Later on, the initial implementation of the hot deployment service was provided to the members of the Globus project team at the University of Chicago and the Argonne National Laboratory, and hot Grid service deployment is currently being integrated into future releases of the Globus toolkit.

Prior to the research this thesis reports on, P2P networking has only been applied for replica location of experimental data in standards-compliant service-oriented Grid middleware. This thesis reports on the first integration of a general purpose P2P communication infrastructure in the same environment. While many research efforts focus on the application of P2P index structures for discovery purposes, this thesis also taps the communication capabilities of the P2P network to enable seamless communication over network partitions in an Internet environment. This feature becomes particularly important in the context of service-oriented Grid middleware, since the web service notification standards emerging from this field require two-way communication. The Internet protocols were chosen for their proven communication capabilities and robustness, but they guarantee the seamless communication capabilities only in a client-server communication pattern with distinct and fixed roles as client and server.

Regarding security threads in a service-oriented ad hoc Grid environment, this thesis has clearly identified a number of intra-node security threats and proposed solutions in a dynamic and changing Grid community. Traditional Grid security infrastructures focus on message level security and access control

with a much more static view on exclusively used resources. However, the need for server consolidation and the current trend to on-demand usage of computing resources will lead to a situation where these intra-node security threats become an imminent risk to a number of applications and need to be dealt with. To counter the threats arising from the dynamic capabilities of an ad hoc Grid, a number of novel service-level isolation techniques were introduced in the field of service-oriented Grid middleware. A wide range of security solutions, ranging from very lightweight approaches with respect to memory utilization through operating system call interposition to rather heavyweight approaches such as complete operating system para-virtualization, have been developed.

A final contribution to the service-oriented Grid computing domain has been made in this thesis by addressing not only the runtime environment, but also sophisticated development support. This thesis presented the first model-driven approach to Grid service development and developer and administrator support in an integrated environment in the service-oriented Grid computing domain. Very positive feedback from a growing external user community suggests that such tools are helpful to lower the entry burden into service-oriented Grid computing and to allow user-friendly development of complex Grid applications. A novel separation of one of the modeling layers - the platform specific layer - into an upper, application specific and a lower, target system specific sub-layer was introduced, allowing to easily target different concrete implementations of the same high-level middleware paradigm.

7.2 Future Work

There are several areas for future work based on the ad hoc Grid middleware presented in the previous chapters. Currently, the MAGE middleware system is in a prototypical implementation state. Therefore, stabilization of the various components and performance optimization is a reasonable first step for future developments. Concrete extensions and future research directions for the individual components of the middleware will be presented in the following.

Peer-to-Peer Integration with the Grid Middleware

At its base, the MAGE system uses a P2P framework to realize a fully decentralized discovery mechanism. The same middleware component is also used for service messaging over network barriers. Tremendous research efforts have been put into using P2P systems for resource discovery while information services have been standardized for the service-oriented Grid and for web services in general. This area would benefit from a consolidation of the efforts and standardization to prevent repeated research and development efforts and to find a suitable solution combining both. Relatively few efforts have been put into the analysis of general

P2P messaging solutions for Grid service communication and especially in the use of P2P networks for efficient transmission of large amounts of binary data. Finally, performance issues related to the local interaction between the higher level Grid service middleware and lower level P2P algorithms and the overhead connected to requiring a client to join a P2P network prior to being able to interact with other nodes offer great potential for future improvements.

Hot Deployment

A central feature of the presented ad hoc Grid middleware is the hot deployment service that allows to introduce new Grid services into the environment without interrupting the running tasks and services of other users. In its current implementation, the service is well suited to handle deployment of pure Java services or services wrapping legacy code that are bound as native libraries. Other Grid applications rely on heavyweight applications that need complex installations on a node. The current solution assumes that deployment of services bearing such dependencies is selectively performed on Grid nodes that already provide the required software. Virtualization techniques such as Xen [23] may offer the ability to deploy such heavyweight applications with an entire operating system instance. There are also emerging standards such as the Web Service Distributed Management specification that may form a basis for inclusion of the HDS functionality and standardization of an architecture and interface for general deployment issues. The exposure of the deployment functionality further enables a new application development style that makes preparation of the bare Grid environment through deployment of required component Grid services possible. A direction of future research may explore the possibilities offered by more sophisticated techniques from areas such as automated planning to this complex task.

Security

Isolation of services and strong separation from other service instances and the underlying Grid resource are the central security issues dealt with in this work. An emphasis is put on lightweight solutions with respect to the memory overhead incurred by the system and solutions also for native legacy parts of a Grid service implementation. The isolation of Grid service instances against each other and the protection of the underlying Grid resource against a dynamically deployed Grid service are the issues that can be dealt with by applying technology currently available. The problem that cannot adequately be addressed with current technology is the protection of a Grid service implementation or the user data against the owner of the underlying Grid resource. Emerging work on the trusted computing platform may enable developers to virtually extend the domain of control and trust of a Grid user to the Grid resource of an unknown untrusted provider.

Many interesting research opportunities will open up with the integration of the technology into service-oriented ad hoc Grid middleware systems such as MAGE.

Virtual Organization Management

A substantial effort has been put into the development of infrastructure support for the formation and management of virtual organizations in the service-oriented Grid. These efforts form the basis for access control and other security related functions of a static Grid environment. Many of the developments in this area are unfortunately rather static and require substantial manual work for the management of virtual organizations. Future work in this direction includes the development of virtual organization management schemes that better suit the needs of a service-oriented ad hoc Grid environment.

Data Handling

The data handling component of the MAGE system enables application and Grid service developers to perform efficient transmission of binary large amounts of binary data while retaining a service-oriented programming model. A benefit of the system is to enable the implementation of complex and sophisticated transmission patterns as a part of the middleware instead of forcing the application developer to embed the data transmission logic in the implementation logic. Research may be put into the identification of common and more complex transmission patterns and their implementation to provide them to users of the middleware. The decoupling of the data transmission logic from the application logic already enables adaptation to changes in the Grid environment. More sophisticated and autonomous mechanisms may be embedded in the ad hoc Grid middleware that globally optimize data transmissions throughout an entire Grid environment.

Grid Process Execution

Currently, the Grid process execution environment of the MAGE system is based on the BPEL standard with additions and small extensions to apply BPEL in the Grid environment. Some of the shortcomings of the current BPEL standard are actively discussed in the BPEL community, therefore future BPEL standards will be more directly be applicable to the Grid. These future changes should be reflected in the MAGE process execution component in order to apply standard tools for process modeling to Grid process creation. A scheduling mechanism is central to the efficient use of a Grid system. The current implementation of the scheduling system integrated into the MAGE system uses a basic strategy. An area for future work in that direction includes the identification and integration of more sophisticated, suitable scheduling mechanisms into the Grid process execution engine. Furthermore, the MAGE system does not enforce usage of a single Grid process execution engine to the entire Grid environment. Therefore,

investigation of distributed and co-scheduling strategies is an interesting research area.

Quality of Service

In its current implementation, the MAGE middleware delivers all functionality and services on a best effort policy. It is, however, desirable to offer service level guarantees to the users of a service-oriented Grid system. The fluctuating ad hoc Grid environment poses special challenges to the realization of quality of service guarantees. Different directions for research open themselves up in this area such as, for example, the analysis of the past behavior of service and resource providers and of clients may be used to predict future behavior. Also recent standardization efforts such as the WS-Agreement standard offer the means to further develop mechanisms for QoS negotiations. Finally, quality requirements and guarantees may be used in other components of the service-oriented ad hoc Grid middleware such as the scheduling system and lead to a better user-experience and utilization of the entire Grid environment.

Grid Development Support

Grid development support is separated into three components: Grid service creation, Grid process creation, interactive debugging and integrated Grid management and visualization tools. Automatic code generation based on modeling of the binding aspects to the underlying Grid middleware are a major benefit of the Grid service creation component. GDT has explicitly been developed to enable adaptation to various service-oriented Grid middleware systems. Due to the limited availability of such systems, currently only mappings to GT4 and the MAGE system are included. An area of future work can be the implementation of additional target system mappings such as a mapping to Unicore/GS [173] once the middleware becomes available in a stable version. Further refinements of the upper layer Grid platform specific meta model may be derived from this mapping. The generated code already introduces a certain degree of separation of concerns into the final service implementations. More possible implementation patterns may be developed to provide greater flexibility and possibly improve the performance of the final application. A number of emerging high level programming toolkits and layers promise abstraction from Grid middleware issues. Support for these interfaces may be included in the GDT. In the area of Grid process creation support, more sophisticated collaboration protocols may be included in the implementation. Additionally, support for the identification and application of common reusable process patterns to the modeling of new patterns may be embedded in the process editor. Also, knowledge mining approaches may be applied to Grid process collections to automatically derive such patterns and best practices. A closer integration of the standard Grid monitoring components and

the interactive visualization modules provided by the GDT may be developed to further assist in the control and management the ad hoc Grid environment. Research in this area may be applied to data reduction and fusion and augmented presentation to the Grid administrator or user. Debugging support for Grid applications should in the future be extended from distributed debugging of Grid services to integrated debugging of entire Grid environments, including capture of the distributed execution state to allow inspection of the distributed state at arbitrary points in time to better understand the dynamic interactions in the highly complex and distributed environment.

List of Figures

Bibliography

[1] N. Abu-Ghazaleh and M. J. Lewis. Differential Deserialization for Optimized SOAP Performance. In *Proc. of the Int'l. Conference for High Performance Computing, Networking, and Storage*, page 21. IEEE Press, 2005.

[2] N. Abu-Ghazaleh, M. J. Lewis, and M. Govindaraju. Differential Serialization for Optimized SOAP Performance. In *Procs. of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 55–64, 2004.

[3] ActiveBPEL, LLC. ActiveBPEL - BPEL execution engine. `http://www.activebpel.org`.

[4] A. Akram, D. Meredith, and R. Allan. Evaluation of BPEL to Scientific Workflows. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 269–274, Singapore, 2006.

[5] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, Vol. 28 (5):749–771, May 2002.

[6] W. Allcock, J. Bester, J. Bresnahan, S. Meder, P. Plaszczak, and S. Tuecke. GridFTP: Protocol Extensions to FTP for the Grid, April 2003. `http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf`.

[7] K. Amin, G. von Laszewski, and A. R. Mikler. Grid Computing for the Masses: An Overview. In *Grid and Cooperative Computing*, pages 464–473, Shanghai, China, 2003.

[8] K. Amin, G. von Laszewski, and A. R. Mikler. Toward an Architecture for ad hoc Grids. In *Proceedings of the International Conference on Advanced Computing and Communications*, India, 2004.

[9] K. Amin, G. von Laszewski, and A. R. Mikler. Quality Assured Ad Hoc Grids. In *Proceedings of the International Conference on Networking and Services*, page 92, Papeete, Tahiti, 2005.

[10] K. Amin, G. von Laszewski, M. Sosonkin, A. R. Mikler, and M. Hategan. Ad Hoc Grid Security Infrastructure. In *Proceedings of the 6th ACM/IEEE International Workshop on Grid Computing*, page 8, Seattle, USA, 2005.

[11] P. Amnuaykanjanasin and N. Nupairoj. The BPEL Orchestrating Framework for Secured Grid Services. In *International Conference on Information Technology: Coding and Computing*, pages 348–353, Las Vegas, USA, 2005.

[12] M. Amoretti, F. Zanichelli, and G. Conte. SP2A: A Service-Oriented Framework for P2P-based Grids. In *MGC '05: Proceedings of the 3rd international Workshop on Middleware for Grid Computing*, pages 1–6, New York, NY, USA, 2005. ACM Press.

[13] N. Andrade, L. Costa, G. Germoglio, and W. Cirne. Peer-to-peer Grid Computing with the Ourgrid Community. In *Proceedings of the 23rd Brazilian Symposium on Computer Networks*, 2005.

[14] N. Andraden, W. Cirne, F. Brasileiro, and P. Roisenberg. Ourgrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 61–68, 2003.

[15] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Knya, M. Mambelli, J. M. Shopf, M. Viljoen, and A. Wilson. GLUE Information Model (Version 1.2), Dec 2005. `http://infnforge.cnaf.infn.it/glueinfomodel/`.

[16] S. Andreozzi, P. Ciancarini, D. Montesi, and R. Moretti. Towards a Metamodeling Based Method for Representing and Selecting Grid Services. In *Lecture Notes in Computer Science, Volume 3270*, pages 78 – 93, 2004.

[17] A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. In *Proc. of the 2nd International Conference on Peer-to-Peer Computing*, pages 33–40. IEEE Computer Society, 2002.

[18] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. C. Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The Design and Implementation of Grid Database Services in OGSA-DAI. *Concurrency and Computation: Practice and Experience*, 17(2-4):357–376, 2005.

[19] Apache Software Foundation. Axis Installation Instructions, 2004. `http://ws.apache.org/axis/java/install.html`.

[20] Apache Software Foundation. Apache Tomcat 5.0, 2005. `http://jakarta.apache.org/tomcat/index.html`.

[21] F. Azzedin and M. Maheswaran. Evolving and Managing Trust in Grid Computing Systems. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1424–1429, Winnipeg, Canada, 2002.

[22] S. Banerjee, S. Basu, S. Garg, S. Garg, S.-J. Lee, P. Mullan, and P. Sharma. Scalable Grid Service Discovery based on UDDI. In *Proc. of the 3rd International Workshop on Middleware for Grid Computing*, pages 1–6, 2005.

[23] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Bolton Landing, USA, 2003. ACM Press.

[24] T. Barth. *Verteilte Lösungsansätze für simulations-basierte Optimierungsprobleme*. Wissenschaftlicher Verlag Berlin, 2001.

[25] J. J. Barton, S. Thatte, and H. F. Nielsen. SOAP Messages with Attachments. W3C Note, 2000.

[26] S. Benkner, I. Brandic, G. Engelbrecht, and R. Schmidt. VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing. In *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 11–18, 2004.

[27] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[28] E. Bertino, P. Mazzoleni, B. Crispo, and S. Sivasubramanian. Towards Supporting Fine-Grained Access Control for Grid Resources. In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pages 59–65, 2004.

[29] M. Box. A new Method of Constrained Optimization and a Comparison with other Methods. *Computer Journal*, 8:42–52, 1965.

[30] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. Towards Quality of Service Support for Grid Workflows. In *Proc. of the European Grid Conference 2005*, pages 661–670, 2005.

[31] A. Brown. An Introduction to Model Driven Architecture Part I: MDA and Todays Systems. *IBM Whitepaper*, pages 1–15, 2004. IBM The Rational Edge.

[32] D. Byrne, A. Hume, and M. Jackson. Grid Services and Microsoft .NET. In *Proc. of UK e-Science All Hands Meeting*, pages 129–136, 2003.

[33] C. Schuler and R. Weber and H. Schuldt and H.-J. Schek. Peer-to-Peer Process Execution with Osiris. In *Proc. of the International Conference on Service Oriented Computing*, LNCS 2910, pages 483–498. Springer-Verlag, 2003.

[34] B. Calder, A. A. Chien, J. Wang, and D. Yang. The Entropia Virtual Machine for Desktop Grids. In *Proc. of the First ACM/USENIX Conference on Virtual Execution Environments*, pages 186–196, Chicago, USA, 2005.

[35] L. Cao, M. Li, H. Rong, and J. Huang. An ontology-based model for grid resource publication and discovery. In *Proc. of the 3rd International Conference on Grid and Cooperative Computing*, pages 448–455, Wuhan, China, 2004.

[36] D. Caromel, A. di Costanzo, D. Gannon, and A. Slominski. Asynchronous peer-to-peer Web services and firewalls. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 6–11, 2005.

[37] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scalable Application-level Anycast for Highly Dynamic Groups. In *Networked Group Communication, Fifth International COST264 Workshop (NGC'2003)*, pages 47–57, 2003.

[38] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays. In *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM2003)*, volume 2, pages 1510–1520, San Francisco, USA, 2003. IEEE Press.

[39] A. Chakravarti, G. Baumgartner, and M. Lauria. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, pages 373–384, 2005.

[40] K. Chang, A. Dasari, H. Madduri, A. Mendoza, and J. Mims. Design of an enablement process for on demand applications. *IBM Systems Journal*, 43(1):190–203, 2004.

[41] K.-M. Chao, M. Younas, N. Griffiths, I. Awan, R. Anane, and C.-F. Tsai. Analysis of Grid Service Composition with BPEL4WS. In *Proc. of the 18th International Conference on Advanced Information Networking and Applications*, pages 284–289, 2004.

[42] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-Like P2P Systems Scalable. In *Proceedings of the 2003 Conference on Applications, Technologies, Ar-chitectures, and Protocols for Computer Communications*, pages 407–418. ACM, 2004.

[43] A. S. Cheema, M. Muhammad, and I. Gupta. Peer-to-peer Ddiscovery of Computational Resources for Grid Applications. In *Proc. of The 6th IEEE/ACM International Workshop on Grid Computing*, pages 179–185, 2005.

[44] L. Chen, K. Candan, J. Tatemura, D. Agrawal, and D. Cavendish. On Overlay Schemes to Support Point-in-Range Queries for Scalable Grid Resource Discovery. In *Proc. of the 5th IEEE International Conference on Peer-to-Peer Computing*, pages 23–30. IEEE Press, 2005.

[45] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language, 2001. `http://www.w3.org/TR/wsdl`.

[46] clusterresources.com. Maui cluster scheduler. `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php`.

[47] clusterresources.com. Torque resource manager. `http://www.clusterresources.com/pages/products/torque-resource-manager.php`.

[48] G. Czajkowski, L. Dayns, and B. Titzer. A Multi-User Virtual Machine. In *Proc. of the USENIX 2003 Annual Technical Conference*, pages 85–98, San Antonio, USA, 2003.

[49] G. Czajkowski, L. Dayns, and M. Wolczko. Automated and Portable Native Code Isolation. Technical report, 2001-96, Sun Labs, 2001.

[50] D-Grid Initiative. Project Website. `http://www.d-grid.de`.

[51] J. Dike. User-Mode Linux. In *Proc. of the 5th Annual Linux Showcase and Conference*, Oakland, USA, 2001.

[52] E. Dodonov, J. Q. Sousa, and H. C. Guardia. GridBox: Securing Hosts from Malicious and Greedy Applications. In *Proc. of the 2nd International Workshop on Middleware for Grid Computing*, pages 17–22, Toronto, Ontario, 2004. ACM.

[53] Eclipse.org. BPEL Project. `http://www.eclipse.org/bpel/`.

[54] eclipse.org. Eclipse Communication Framework (ECF). `http://www.eclipse.org/ecf`.

[55] eclipse.org. Eclipse Modeling Framework. `http://www.eclipse.org/emf`.

[56] eclipse.org. Web Tools Project.
`http://eclipse.org/webtools`.

[57] EGEE. Enabling Grids for eScience in Europe: Executive Summary, 2004.
`http://egee-intranet.web.cern.ch/egee-intranet`.

[58] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price. Grid Service Orchestration using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3-4):283–304, 2005.

[59] W. Emmerich and A. L. Wolf, editors. *Component Deployment, Second International Working Conference, CD 2004, Edinburgh, UK, May 20-21, 2004, Proceedings*, volume 3083 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[60] R. Ewerth, T. Friese, M. Grube, and B. Freisleben. Grid Services for Distributed Video Cut Detection. In *Proc. of the 6th IEEE Int. Symposium on Multimedia Software Engineering*, pages 164–168, Miami, USA, 2004. IEEE Press.

[61] R. Ewerth, J. Gllavata, M. Gollnick, F. Mansouri, E. Papalilo, R. Sennert, J. Wagner, B. Freisleben, and M. Grauer. Methoden und Werkzeuge zur rechnergestützten medienwissenschaftlichen Analyse. *Siegener Periodicum zur Internationalen Empirischen Literaturwissenschaft*, 20, H. 2:306–320, 2003.

[62] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Junior, and H.-L. Truong. ASKALON: A Tool Set for Cluster and Grid Computing. *Concurrency and Computation: Practice and Experience*, 17(2-4), 2005. http://dps.uibk.ac.at/askalon/.

[63] T. Fahringer, S. Pllana, and A. Villazon. AGWL: Abstract Grid Workflow Language. In *Proc. of the 4th International Conference on Computational Science*, volume 3038 of *LNCS*, pages 42–49, Krakow, Poland, June 2004. Springer-Verlag.

[64] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. ASKALON: A Grid Application Development and Computing Environment. In *6th International Workshop on Grid Computing (Grid 2005)*, pages 10–20, Seattle, USA, 2005. IEEE Press.

[65] T. Fahringer, J. Qin, and S. Hainzer. Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In *Proc. of the IEEE International Symposium on Cluster Computing and the Grid*, pages 676–685, Cardiff, UK, 2005. IEEE Computer Society Press.

[66] M. Fleury and F. Reverbel. The JBoss extensible server. In M. Endler and D. Schmidt, editors, *Proceedings of the 2003 ACM / IFIP / USENIX International Middleware Conference*, pages 344–373, 2003.

[67] F. Flore. MDA: The Proof is in Automating Transformations Between Models. In *OptimalJ White Paper*, pages 1–4, 2003.

[68] I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savvy, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture, Version 1.0. Whitepaper GGF, 2004.

[69] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, pages 118–128, 2003.

[70] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[71] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, pages 1–31, 2002.

[72] G. Fox, S. Pallickara, and S. Parastatidis. Towards Flexible Messaging for SOAP Based Services. In *Proc. of the IEEE/ACM Supercomputing Conference*, page 8, Pittsburg, USA, 2004.

[73] G. Fox, S. Pallickara, and X. Rao. A Scaleable Event Infrastructure for Peer to Peer Grids. In *Proc. of the ACM Java Grande International Symposium on Computing in Object-oriented Parallel Environments Conference*, pages 66–75, Seatlle, USA, 2002.

[74] P. Fraternali and P. Paolini. Model-Driven Development of Web Applications: The Autoweb System. In *ACM Transactions on Information Systems, Vol. 28*, pages 323–382, 2000.

[75] N. Freed, N. Borenstein, K. Moore, J. Klensin, and J. Postel. RFC 2045-2049: Multipurpose Internet Mail Extensions (MIME), 1996.

[76] freepastry.org. FreePastry. `http://freepastry.org`.

[77] T. Friese, B. Freisleben, S. Rusitschka, and A. Southall. A Framework for Resource Management in Peer-to-Peer Networks. In *Proceedings of the Int. Conference NetObjectDays*, LNCS 2591, pages 4–21, Erfurt, Germany, 2002. Springer-Verlag.

[78] T. Friese, J. P. Müller, and B. Freisleben. Self-Healing Execution of Business Processes Based on a Peer-to-Peer Service Architecture. In *Proceedings of the 18th Int. Conference on Architecture of Computing Systems*, number 3432 in LNCS, pages 108–123, Innsbruck, Austria, 2005. Springer-Verlag.

[79] T. Friese, J. P. Müller, and B. Freisleben. Integrating Peer-to-Peer Technology into a Web Service Environment. In *Multi-Konferenz Wirtschaftsinformatik Workshop Betriebliche Anwendungen des P2P und Grid Computing*, Passau, Germany, 2006.

[80] T. Friese, J. P. Müller, M. Smith, and B. Freisleben. A Robust Business Resource Management Framework Based on a Peer-to-Peer Infrastructure. In *Proceedings of the 7th International IEEE Conference on E-Commerce Technology*, pages 215–222, Munich, Germany, 2005. IEEE Press.

[81] T. Friese, M. Smith, and B. Freisleben. Hot Service Deployment in an Ad Hoc Grid Environment. In *Proceedings of the 2nd Int. Conference on Service-Oriented Computing*, pages 75–83, New York, USA, 2004. ACM Press.

[82] T. Friese, M. Smith, and B. Freisleben. GDT: A Toolkit for Grid Service Development. In *Proceedings of the 3rd International Conference on Grid Service Engineering and Management*, pages 131–148, Erfurt, Germany, 2006.

[83] T. Friese, M. Smith, and B. Freisleben. The Service-Oriented Ad Hoc Grid. In *Barth T., Schüll A. (eds): Grid Computing*, pages 143–191. Vieweg Verlag, 2006.

[84] T. Friese, M. Smith, B. Freisleben, J. Reichwald, T. Barth, and M. Grauer. Collaborative Grid Process Creation Support in an Engineering Domain. In *Proc. of the International Conference on High Performance Computing*, page (to appear), Bangalore, India, 2006. Springer-Verlag.

[85] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington. ICENI: An Open Grid Services Architecture Implemented with Jini. In *Proc. of UK e-Science All Hands Meeting*, pages 703–120, 2003.

[86] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington. Implementations of a Service-Oriented Architecture on Top of Jini, JXTA and OGSI. In *Proc. Second Across Grids Conference*, volume 3165 of *LNCS*, pages 90–99. Springer-Verlag, 2004.

[87] A. Ganguly, A. Agrawal, P. Boykin, and R. Figueiredo. IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing. In *Proc.*

of the 20th International Parallel and Distributed Processing Symposium, pages 1–10, Rhodes Island, Greece, 2006. IEEE Press.

[88] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. WOW: Self-Organizing Wide Area Overlay Networks of Virtual Workstations. In *Proc. of the 15th IEEE International Symposium on High Performance Distributed Computing*, page (to appear), Paris, France, 2006.

[89] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski. *Grid Computing: Making the Global Infrastructure a Reality*, chapter Grid Web Services and Application Factories. Wiley, 2003.

[90] S. Genaud and C. Rattanapoka. A Peer-to-Peer Framework for Robust Execution of Message Passing Parallel Programs on Grids. In *Proc. of the 12th European PVM/MPI Users Group Meeting*, number 3666 in LNCS, pages 276–284, Sorento, Italy, 2005. Springer-Verlag.

[91] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legr, C. Loomis, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-Enabling Medical Image Analysis. In *Proc. of the BioGrid Workshop at CCGrid 2005*, pages 487–495, Cardiff, UK, 2005. IEEE Computer Society.

[92] T. Glatard, J. Montagnat, and X. Pennec. Grid-Enabled Workflows for Data Intensive Applications. In *Proc. of the 18th IEEE Symposium on Computer-Based Medical Systems*, pages 537–542, Dublin, Ireland, 2005. IEEE Computer Society.

[93] A. Gokhale and B. Natarajan. Composing and Deploying Grid Middleware Web Services Using Model Driven Architecture. In *Lecture Notes in Computer Science, Volume 2519*, pages 633 – 649, 2002.

[94] Y. Gong, W. Li, Y. Sun, and Z. Xu. A C/S and P2P Hybrid Resource Discovery Framework in Grid Environments. In *Proc. of the International Conference on Parallel Processing*, pages 261–268, 2005.

[95] J. Han and D. Park. A Lightweight Personal Grid Using a Supernode Network. In *Proceedings of the 3rd International IEEE Conference on Peer-to-Peer Computing*, pages 168–175, 2003.

[96] F. Heine, M. Hovestadt, and O. Kao. Towards Ontology-driven P2P Grid Resource Discovery. In *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 76–83, 2004.

[97] S. Heinzl, M. Mathes, T. Friese, M. Smith, and B. Freisleben. Flex-SwA: Flexible Exchange of Binary Data Based on SOAP Messages with Attachments. In *Proc. of the IEEE International Conference on Web Services*, pages 3–10, Chicago, USA, 2006. IEEE Press.

[98] Z. Huang, C. He, L. Gu, and J. Wu. On-Demand Service in Grid: Architecture, Design and Implementation. In *11th International Conference on Parallel and Distributed Systems - Workshops*, pages 674–678, 2005.

[99] A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. In *Proc. of the International Workshop on Grid Computing*, pages 51–62, Denver, USA, 2001.

[100] A. Iamnitchi, I. Foster, and D. Nurmi. A Peer-to-Peer Approach to Resource Location in Grid Environments. In *Proc. of the 11th Symposium on High Performance Distributed Computing*, page 419, 2002.

[101] IBM Global Services. FAQs: IBM e-Business On Demand, 2002. `http://www.ibm.com/services/ondemand/files/Q\&A.pdf`.

[102] W. Jie, T. Hung, S. Turner, and W. Cai. Architecture Model for Information Service in Large Scale Grid Environments. In *Proc. of the 6th IEEE International Symposium on Cluster Computing and the Grid.*, pages 107–114, Singapore, 2006. IEEE Press.

[103] M. Jordan, L. Dayns, G. Czajkowski, M. Jarzab, and C. Bryce. Scaling J2EE(TM) Application Servers with the Multi-Tasking Virtual Machine. Technical report, SMLI TR-2004-135, Sun Microsystems Laboratories , 2004.

[104] F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, volume 3844 of *LNCS*, pages 128–138, Montego Bay, Jamaica, 2005. Springer-Verlag.

[105] P.-H. Kamp and R. N. M. Watson. Jails: Confining the omnipotent root, 2000. `http://docs.freebsd.org/44doc/papers/jail/jail.html`.

[106] K. Karasavvas, M. Antonioletti, M. Atkinson, N. C. Hong, T. Sugden, A. Hume, M. Jackson, A. Krause, and C. Palansuriya. Introduction to OGSA-DAI Services. In *Lecture Notes in Computer Science*, volume 3458, pages 1–12. Springer-Verlag, 2005.

[107] M. Koh, J. Song, L. Peng, and S. See. Service Registry Discovery using GridSearch P2P Framework. In *Proc. of the 6th IEEE International*

Symposium on Cluster Computing and the Grid Workshops, pages 11–16, Singapore, 2006. IEEE Press.

[108] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@home - Massively Distributed Computing for SETI. *Computing in Science and Engineering*, 3(1):79, 2001.

[109] X. Lai and J. L. Massey. A Proposal for a New Block Encryption Standard. *Lecture Notes in Computer Science*, 473:389+, 1991.

[110] G. Laschet, J. Neises, and I. Steinbach. Micro- Macrosimulation of Casting Processes. 4^{ieme} *école d'été de Modélisation numérique en thermique*, C8:1–42, 1998.

[111] Lawrence Berkeley National Laboratory. pyOGSI, 2003. `http://www-itg.lbl.gov/gtg/projects/pyOGSI/`.

[112] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. IETF RFC 4122, 2005. `http://www.ietf.org/rfc/rfc4122.txt`.

[113] F. Leymann. Choreography for the Grid: Towards Fitting BPEL to the Resource Framework. In *Concurrency and Computation: Practice and Experience*. John Wiley & Sons, Ltd., 2005 (online).

[114] J. Li and S. Vuong. A Semantics-Based Routing Scheme for Grid Resource Discovery. In *Proc. of the First International Conference on e-Science and Grid Computing*, pages 438–445, 2005.

[115] S. Lohr. IBM Making a Commitment to the Next Phase of the Internet, 2001. `http://www.nytimes.com/2001/08/02/technology/02BLUE.html?pagewanted=print`.

[116] D. Manset, R. McClatchey, F. Oquendo, and H. Verjus. A Model-driven Approach for Grid Services Engineering. In *Proc of the 18th international conference on Software and Systems Engineering and Applications*, pages 51–58, Paris, France, 2005.

[117] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*, number 3387 in LNCS, pages 26–42, San Diego, USA, 2004. Springer-Verlag.

[118] M. Mathes, S. Heinzl, T. Friese, and B. Freisleben. Enabling Post-Invocation Parameter Transmission in Service-Oriented Environments. In *Proc. of the International Conference on Networking and Services*, page 55, Silicon Valley, USA, 2006. IEEE Press.

[119] A. S. McGough, W. Lee, and J. Darlington. ICENI II Architecture. In *UK e-Science All Hands Meeting*. Nottingham, UK, 2005.

[120] S. Mizuta and R. Huang. Automation of Grid Service Code Generation with AndroMDA for GT3. In *Proc. of the 19th International Conference on Advanced Information Networking and Applications (AINA 2005)*, pages 417–420, 2005.

[121] H. Morohoshi and R. Huang. A User-friendly Platform for Developing Grid Services over Globus Toolkit 3. In *Proc. of the 11th International Conference on Parallel and Distributed Systems*, pages 668–674, Fuduoka, Japan, 2005. IEEE Computer Society.

[122] J. Mukerji and J. Miller. Overview and guide to OMG's architecture. *IBM Whitepaper*, pages 1–62, 2001. IBM The Rational Edge.

[123] R. Mulye. Modeling Web Services using UML/MDA, 2005. `http://lsdis.cs.uga.edu/~ranjit/academic/essay.pdf`.

[124] M. Murshed and R. Buyya. Using the GridSim Toolkit for Enabling Grid Computing Education. In *Proc. of the Int. Conf. on Communication Networks and Distributed Systems Modeling and Simulation*, pages 18–24, 2002.

[125] V. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan. Harmony: A Desktop Grid for Delivering Enterprise Computations. In *Proceedings. Fourth International Workshop on Grid Computing*, pages 25–33, 2003.

[126] J. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–311, 1965.

[127] T. Nguyen and V. Selmin. Collaborative Multidisciplinary Design in Virtual Environments. In *Proceedings of the 10th International Conference on CSCW in Design.*, pages 420–425, Nanjing, China, 2006.

[128] S. Nickolas. IBM Framework for e-business: Application hosting services. Developerwerks, IBM Corporation, 1999. `http://www-136.ibm.com/developerworks`.

[129] H. F. Nielsen, H. Sanders, R. Butek, and S. Nash. Direct Internet Message Encapsulation (DIME), 2002. `http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt`.

[130] R. G. nmo, D. Skogan, I. Solheim, and J. Oldevik. Model-driven Web Services Development. In *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE04)*, pages 633 – 649, 2004.

[131] OASIS. Introduction to UDDI Important Features and Functional Concepts. Technical White Paper, pp. 1-11. `http://uddi.org/pubs/uddi-tech-wp.pdf`.

[132] OASIS. Web Service Resource Framework, 2004. `http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=wsrf`.

[133] Oasis WSBPEL TC. Web Services Business Process Execution Language Version 2.0 - draft, December 2005.

[134] Object Management Group, Inc. MDA Guide 1.0.1, omg/2003-06-01, June 2003.

[135] OGSI::Lite. Perl Grid Services, 2003. `http://www.sve.man.ac.uk/Research/AtoZ/ILCT`.

[136] S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In *Proc. of the ACM/IFIP/USENIX International Middleware Conference*, volume 2672 of *LNCS*, pages 41–61, Rio de Janeiro, Brazil, 2003. Springer-Verlag.

[137] E. Papalilo, T. Friese, M. Smith, and B. Freisleben. Trust Shaping: Adapting Trust Establishment and Management to Application Requirements in a Service-Oriented Grid Environment. In *Proc. of the 4th International Conference on Grid and Cooperative Computing*, pages 47–58, Beijing, China, 2005. Springer-Verlag.

[138] prefuse.org. Prefuse Information Visualization Toolkit. `http://prefuse.org`.

[139] Project JXTA. Java Programmers Guide v 2.0, 2003.

[140] N. Provos. Improving Host Security with System Call Policies. In *Proc. of the 12th USENIX Security Symposium*, pages 257–272, Washington, USA, 2003.

[141] L. Qi, H. Jin, I. Foster, and J. Gawor. HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4. Submitted for publication, 2006.

[142] R. Radhakrishnan and M. Wookey. Model Driven Architecture Enabling Service Oriented Architectures. In *Whitepaper SUN Microsystems*, pages 1 – 13, 2004.

[143] M. A. Rappa. The Utility Business Model and the Future of Computing Services. *IBM Systems Journal*, 43(1):32–42, 2004.

[144] T. Reenskaug. MODELS - VIEWS - CONTROLLERS. Technical Note, Xerox PARC, 1979. A scanned version on: `http://heim.ifi.uio.no/~trygver/mvc/index.html`.

[145] M. Roussopoulos, M. Baker, D. Rosenthal, T. Guili, P. Maniatis, and J. Mogul. 2 P2P or not 2 P2P? In *Proceedings of the 3rd international Workshop on Peer-to-Peer Systems*, pages 33–44, San Diego, USA, 2004. Springer-Verlag.

[146] S. Rusitschka and A. Southall. The Resource Management Framework: A System for Managing Metadata in Decentralized Networks Using Peer-to-Peer Technology. In *Proc. of the 1st Int. Workshop on Agents and Peer-to-Peer Computing*, pages 144–149, Bologna, Italy, 2002. Springer-Verlag.

[147] C. Schmidt and M. Parashar. A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web Journal*, 7(2):211–229, 2004.

[148] B. Seshasayee, K. Schwan, and P. Widener. SOAP-binQ High-Performance SOAP with Continuous Quality Management. In *Proc. of the 24th International Conference on Distributed Computing Systems*, pages 158–165, 2004.

[149] S. Shivle, H. Siegel, A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco. Mapping of Subtasks with Multiple Versions in a Heterogeneous Ad Hoc Grid Environment. *Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pages 380–387, 2004.

[150] K. Shudo, Y. Tanaka, and S. Sekiguchi. P3: P2P-based Middleware Enabling Transfer and Aggregation of Computational Resources. In *Proc. of the 5th IEEE International Symposium on Cluster Computing and the Grid*, pages 259–266, 2005.

[151] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer. GLARE: A Grid Activity Registration, Deployment and Provisioning Framework. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, pages 52–64, Washington, DC, USA, 2005. IEEE Computer Society.

[152] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf*, page 111, 2004.

[153] M. Smith, T. Friese, M. Engel, and B. Freisleben. Countering Security Threats in Service-Oriented On-Demand Grid Computing Using Sandboxing and Trusted Computing Techniques. *Journal of Parallel and Distributed Computing*, pages 1189–1204, 2006.

[154] M. Smith, T. Friese, M. Engel, B. Freisleben, G. Koenig, and W. Yurcik. Security Issues in On-Demand Grid and Cluster Computing. In *Proc. of the 6th IEEE International Symposium on Cluster Computing and the Grid Workshops*, pages 24–37, Singapore, 2006. IEEE Press.

[155] M. Smith, T. Friese, and B. Freisleben. Intra-Engine Service Security for Grids Based on WSRF. In *Proceedings of the 2004 International Conference on Cluster Computing and Grid*, pages 644–653, Cardiff, UK, 2004. IEEE Press.

[156] M. Smith, T. Friese, and B. Freisleben. Towards a Service-Oriented Ad Hoc Grid. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing*, pages 201–209, Cork, Ireland, 2004. IEEE Press.

[157] M. Smith, T. Friese, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications. In *Proc. of the International Conference on Internet and Web Applications and Services, Guadeloupe*, pages 139–146, Guadeloupe, French Guyana, 2006. IEEE Press.

[158] M. Smith, T. Friese, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications. In *Barth T., Schüll A. (eds.): Grid Computing*, pages 191–210. Vieweg Verlag, 2006.

[159] B. Sotomayor. The Globus Service Build Tools Project. `http://gsbt.sourceforge.net/`.

[160] H. D. Sterck, R. Markel, and R. Knight. A Lightweight, Scalable Grid Computing Framework for Parallel Bioinformatics Applications. In *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, pages 251–257. IEEE Press, 2005.

[161] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM Special Interest Group on Data Communication 2001 Technical Conference*, pages 149–160, San Diego, USA, 2001. ACM Press.

[162] Sun Microsystems, Inc. Java Platform Debugger Architecture (JPDA). http://java.sun.com/products/jpda/.

[163] Sun Microsystems, Inc. Jini Network Technology, 1999. http://www.sun.com/software/jini.

[164] Sun Microsystems, Inc. Java Native Interface Specification, 2003.

[165] Sun Microsystems, Inc. Java Specification Request 121: Application Isolation API Specification, 2003.

[166] SWsoft. OpenVZ, 2006. http://openvz.org/.

[167] D. Talia and P. Trunfio. A P2P Grid Services-Based Protocol: Design and Evaluation. In *Proc. of the 10th International Euro-Par Conference*, number 3149 in LNCS, pages 1022–1031, Pisa, Italy, 2004.

[168] D. Talia and P. Trunflo. Toward a Synergy Between P2P and Grids. *Internet Computing*, 7(4):94–95, 2003.

[169] The Apache Software Foundation. The Apache Ant Project, 2004. http://ant.apache.org.

[170] The Globus Alliance. GT Data Management: Reliable File Transfer (RFT), November 2005.

[171] The World Wide Web Consortium. Simple Object Access Protocol (SOAP), 2003. http://www.w3.org/TR/soap/.

[172] D. Thomas. MDA: Revenge of the Modelers or UML Utopia? In *IEEE Software, May/June* , pages 15–17, 2004.

[173] UniGrids EU Project. Unicore/GS. http://www.unigrids.org.

[174] G. von Laszewski, E. Blau, M. Bletzinger, J. Gawor, P. Lane, S. Martin, and M. Russell. Software, Component, and Service Deployment in Computational Grids. In *IFIP/ACM Working Conference on Component Deployment*, volume 2370 of *LNCS*, pages 244–256, Berlin, Germany, 2002. Springer-Verlag.

[175] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001.

[176] G. S. Wasson, N. Beekwilder, M. M. Morgan, and M. Humphrey. OGSI.NET: OGSI-Compliance on the .NET Framework. In *4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 648–655, Chicago, USA, 2004. IEEE Press.

[177] Wikipedia. Entry on Gnutella. `http://en.wikipedia.org/wiki/Gnutella`.

[178] S. Woyak, H. Kim, J. Mullins, and J. Sobieszczanski-Sobieski. A Web Centric Architecture for Deploying Multi-Disciplinary Engineering Design Processes. In *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, New York, USA, 2004.

[179] Q. Xia, W. Wang, and R. Yang. A Fully Decentralized Approach to Grid Service Discovery Using Self-organized Overlay Networks. In *Proc. of the European Grid Conference*, number 3470 in LNCS, pages 164–172. Springer-Verlag, 2005.

[180] Q. Xia, R. Yang, W. Wang, and D. Yang. Fully Decentralized DHT based Approach to Grid Service Discovery using Overlay Networks. In *Proc. of the 5th International Conference on Computer and Information Technology*, pages 1140–1045, Shanghai, China, 2005. IEEE Computer Society.

[181] G. Xue, G. E. Pound, and S. J. Cox. Performing Grid Computation with Enhanced Web Service and Service Invocation Technologies. In *Proc. of the Int'l. Conference on Computational Science, LNCS 2659*, pages 297–306. Springer-Verlag, 2003.

[182] Y. Ying, Y. Huang, and D. W. Walker. A Performance Evaluation of Using SOAP with Attachments for e-Science. In *Proc. of the UK e-Science All Hands Meeting*, pages 796–803, 2005.

[183] J. Yu and R. Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. Technical Report GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourn, Australia, 2005.

[184] J. Yu and R. Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *ACM SIGMOD Record*, 34:44–49, 2005.